Communications
INTEROP 7

NatStar on GCOS 7

Subject: This manual explains how to use NatStar, and how to develop NatStar TPRs (Transaction Processing Routines) to run under TDS on GCOS 7.

Special Instructions: Revision 01 of this manual is valid for users of NatStar 7 V2.

Revision 00 remains valid for users of NatStar 7 V1.2.

Software Supported: NatStar 7 V2

NatStar 2.15

Software/Hardware required: DPS 7000 with GCOS 7, TDS, OPEN 7, and NatStar.

Date: March 1999

Bull S.A.
CEDOC
Atelier de reprographie
34, rue du Nid de Pie BP 428
49004 ANGERS Cedex 01
FRANCE

Bull HN Information Systems Inc.
Publication Order Entry
FAX: (978) 294-7411
MA30/865A
Technology Park
Billerica, MA 01821
U.S.A.

47 A2 90US Rev01

Copyright © Bull S.A., 1998, 1999

Bull acknowledges the rights of proprietors of trademarks mentioned herein.

Your suggestions and criticisms concerning the form, contents and presentation of this manual are invited.
A form is provided at the end of this manual for this purpose.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical or otherwise without the prior written permission of the publisher.

Bull disclaims the implied warranties of merchantability and fitness for a particular purpose and makes no express warranties except as may be stated in its written agreement with and for its customer. In no event is Bull liable to anyone for any indirect, special, or consequential damages.
The information and specifications in this document are subject to change without notice.
Consult your Bull Marketing Representative for product or service availability.

47 A2 90US Rev01

Preface

Scope and Objectives

This document deals with the aspects of client/server development that are unique to the use of GCOS 7 as a server system.

These aspects include:

- Using NatStar to generate COBOL-85 code to be executed on GCOS 7. Transferring the generated COBOL-85 code to GCOS 7, then compiling and linking it for use with TDS or another GCOS 7 environment.
- Configuring the TDS environment(s).
- The information contained in this document is intended for use by developers of client applications and TDS server applications, and by GCOS 7 System Administrators.

Intended Readers

The information contained in this document is intended for use by developers of client applications and TDS server applications, and by GCOS 7 System Administrators.

Prerequisites

The reader of this document is assumed to be familiar with NatStar and with the conventional development of Transaction Processing Routines (TPRs) to run under TDS on GCOS 7.

The document focuses on the linking of the two environments and using them effectively to create GCOS 7-based client/server applications.

The reader is also assumed to be generally familiar with the networking software used with OPEN 7 on GCOS 7.

Structure

This manual is structured as follows:

Chapter 1	Introduces the Bull/NatStar product features, and tells you how to use the documentation.
Chapter 2	Offers an overview of client/server concepts, describes the types of client/server configurations supported by Bull/NatStar, and explains the options available for using Bull/NatStar.
Chapter 3	Discusses topics to be considered in planning for the installation of Bull/NatStar.
Chapter 4	Discusses design aspects that are important during high-level design of a Bull/NatStar client/server application. It concludes with a step-by-step guide to the implementation of a client/server application which serves as an introduction to the more detailed discussion of implementation in chapter 5.
Chapter 5	Describes how to implement client/server applications that operate on PC or UNIX client systems and a GCOS 7 TDS server system. The focus in this section is on how to implement NSCOM7 TPRs.
Chapter 6	Describes the use of NatStar in a GCOS 7 batch environment.
Chapter 7	Describes ORACLE Support in TDS/BATCH Mode.
Chapter 8	Describes how to use the NSCOM Light (NS02OP7G).
Chapter 9	Describes how to use the COBOL Subroutines from the NatStar NCL Language.
Appendix A	An introduction to NatStar development workstation sessions and to appendices B through F which contain a simple example of a session in which the NatStar development workstation is used to develop a GCOS 7/TDS-based client/server application.
Appendix B	Step 1: Developing the basic CONCAT server application.
Appendix C	Step 2: Developing the client application (user-interface window and supporting logic) using the CONCAT server application.

Appendix D	Step 3: Defining GCOS 7 configurations that enable GCOS to be used as a NatStar server.
Appendix E	Step 4: Partitioning the application into client/server parts: - Introducing CONNECT and DISCONNECT phases - Introducing access UFAS files.
Appendix F	Step 5: Generating COBOL-85 language source for the server logic.
Appendix G	Contains the full listings of the NCL sequences shown only partially in the screens in Appendix B.
Appendix H	Gives the NatStar TP Error Codes.
Appendix I	Gives the Error Codes for the Socket Interface on PC.
Glossary	The Glossary defines selected terms used in this manual.
Index	The Index provides direct access to the information in this manual.

Bibliography The following documents include information that is related to the Bull/NatStar environment.

Bull/NatStar Documentation

Bull/NatStar Software Release Bulletin (GCOS 7-GX-INT-974)

TDS Documentation

TDS COBOL Programming Guide *47 A2 33UT*
TDS Administrator's Guide *47 A2 32UT*

Language Documentation

COBOL-85 User's Guide *47 A2 06UL*
COBOL-85 Reference Manual *47 A2 05UL*
LINKER User's Guide *47 A2 10UP*

OPEN 7 Documentation

OPEN 7 User's Guide *47 A2 80US*
OPEN 7 Administrator's Guide *47 A2 81US*
Interoperability Software Installation 7 ..... *47 A2 56UU*

General GCOS 7 Documentation

GCOS 7 Messages and Return Codes Directory *47 A2 10UJ*
PCF User's Guide *47 A2 15UP*
UFAS Extended User's Guide *47 A2 04UF*

FCP 7 Documentation

FCP 7 Software Release Bulletin *47 A2 95UC*

Nat Systems Documentation

NatStar Overview ..... *(NSR202OVE01001)*
NatStar Getting Started ..... *(NSR210STA01001)*
NatStar Installation and User Manual -- Adaptable Development Environment ..... *(NSR210ADE01001)*
NatStar User Manual -- Graphical Builder, Process Modeling, Information Modeling ..... *(NSR210USE01001)*
NatStar Development Guide -- Client-Server ..... *(NSR202C_S01001)*
NatStar Development Guide -- Graphical Builder, Process Modeling, Information Modeling ..... *(NSR210DVP01001)*
NatStar Language Reference -- NCL ..... *(***201NCL01001)*
NatStar User Guide and Programming Manual -- Communication Services ..... *(NSR201COM01001)*
NatStar Programming Interface Libraries -- Volume 1 ..... *(***201LV101001)*
NatStar Programming Interface Libraries -- Volume 2 ..... *(***201LV201001)*
NatStar Programming Interface Libraries -- Volume 3 ..... *(NSR210LV301001)*
NatStar Programming Interface -- Database Access ..... *(***201DBA01001)*
NatStar Application Porting Guide ..... *(NSR202POR00001)*
NatStar Version Management User Manual ..... *(NSR210VCM_01001)*
NatStar Report Manual ..... *(NSR210REP01001)*
NatStar CDAM Reference and Programming Guide ..... *(NSR210CDA01001)*

Syntax Notation Not applicable.

Delivery Conditions Not applicable.

Table of Contents

1. Overview

1.1 Purpose of this Document 1-2

1.2 Features and Benefits of Bull/NatStar 1-3

1.3 A Guide to the Use of this Document 1-6

2. Bull/NatStar in the Client/Server Environment

2.1 The Client/Server Architecture 2-2

2.1.1 Basic Client/Server Concepts 2-2

2.1.2 Using Remote Procedure Call (RPC) 2-3

2.1.3 Client/Server Configurations 2-5

2.2 Bull/NatStar Support of the Client/Server Architecture 2-7

2.2.1 Using Bull/NatStar To Implement Client/Server Applications 2-8

2.2.1.1 PC Clients and GCOS 7 Server Using NSCOM7 2-8

2.2.1.2 Launching the NSCOM Daemon on OPEN 7 2-9

2.2.1.3 Launching and Administering NSERV7 2-10

2.3 Options for Using Bull/NatStar 2-14

2.3.1 NatStar TPRs and Conventional TPRs 2-14

2.3.2 Using the Full Bull/NatStar Environment 2-15

2.3.3 Using NatStar Independently 2-16

3. Planning for the Bull/NatStar Environment

3.1 Planning for Required Software and Hardware 3-1

3.1.1 GCOS 7 Required and Optional Software and Hardware 3-2

3.1.2 NatStar Development Workstation Required Software 3-2

3.2 Configuring TDS 3-2

4. Designing Client/Server Applications

4.1 Ensuring Extendibility ... 4-2

4.2 Achieving Satisfactory Performance ... 4-4

4.2.1 Minimizing the Number of Client/Server Interactions ... 4-4

4.2.2 Minimizing the Volume of Data Transferred ... 4-5

4.2.3 Optimizing Network Performance ... 4-5

4.3 Maintaining Data Integrity ... 4-6

4.3.1 Combining Service Requests from the Client ... 4-7

4.3.2 Avoiding Lost Updates ... 4-8

4.3.3 Updating Multiple Databases ... 4-9

4.4 Implementing Security Controls ... 4-11

4.5 Step-by-Step Implementation Guide ... 4-13

5. Implementing Client/Server Applications

5.1 The NSCOM Environment ... 5-2

5.1.1 The Bull/NatStar Development Environment ... 5-2

5.1.1.1 Using the NatStar Development Workstation ... 5-2

5.1.1.2 Using the TDS Environment ... 5-4

5.1.1.3 Using the NSCOM Communication Link ... 5-5

5.1.2 NatStar TPR Structure and Processing Flow ... 5-7

5.2 Special Considerations when Developing a NatStar TPR ... 5-10

5.3 Developing a NatStar TPR ... 5-13

5.3.1 Using UFAS File Access APIs in NCL ... 5-13

5.3.1.1 NSUFAS_LAST_ERROR ... 5-13

5.3.1.2 NSUFAS_LAST_ERRMSG ... 5-15

5.3.1.3 NSUFAS_GET_FILE_HANDLE ... 5-15

5.3.1.4 NSUFAS_READ ... 5-16

5.3.1.5 NSUFAS_READ_INDEXED ... 5-18

5.3.1.6 NSUFAS_READ_RELATIVE ... 5-20

5.3.1.7 NSUFAS_READ_COND ... 5-22

5.3.1.8 NSUFAS_READ_SECKEY ... 5-23

5.3.1.9 NSUFAS_UPDATE ... 5-23

5.3.1.10 NSUFAS_UPDATE_INDEXED ... 5-25

5.3.1.11 NSUFAS_UPDATE_RELATIVE ... 5-27

5.3.1.12 NSUFAS_WRITE ... 5-29

5.3.1.13 NSUFAS_WRITE_INDEXED ... 5-31

5.3.1.14 NSUFAS_WRITE_RELATIVE ... 5-33

5.3.1.15 NSUFAS_START_INDEXED ... 5-35

5.3.1.16 NSUFAS_START_RELATIVE ... 5-37

5.3.1.17 NSUFAS_START_BEGIN ... 5-39
5.3.1.18 NSUFAS_START_SECKEY ... 5-39
5.3.1.19 NSUFAS_DELETE ... 5-40
5.3.1.20 NSUFAS_DELETE_INDEXED ... 5-41
5.3.1.21 NSUFAS_DELETE_RELATIVE ... 5-42
5.3.2 Using Packed-Decimal Data Type in NCL ... 5-44
5.3.3 Handling Data-Type Incompatibilities ... 5-47
5.3.3.1 NCL Data Types that Map to COBOL Data Types ... 5-47
5.3.3.2 Causes of Data-Type Incompatibility ... 5-48
5.3.3.3 Passing Values to a Conventional Subroutine ... 5-49
5.3.3.4 Returning Values from a Conventional Subroutine ... 5-50
5.3.3.5 Converting Data Types Using COBOL Move Statements ... 5-50
5.3.3.6 Converting Data Types Using NCL Data-Conversion Routines ... 5-51
5.3.4 Using Native Code ... 5-56
5.3.5 Calling A Conventional Subroutine ... 5-58
5.3.5.1 Restrictions on Message Receive/Send Operations ... 5-60
5.3.5.2 Other Restrictions ... 5-61
5.3.6 Handling Errors ... 5-61
5.3.6.1 Detecting and Reporting Server and Communication Errors ... 5-62
5.3.6.2 Error Analysis and Resolution ... 5-67
5.4 Generating Server Source Code ... 5-70
5.4.1 Background Information ... 5-70
5.4.2 Generating COBOL Source Code ... 5-90
5.4.2.1 COBOL Generator Output ... 5-92
5.4.2.2 COPY Files Provided ... 5-92
5.5 Compiling a NatStar TPR ... 5-93
5.5.1 Compiling NatStar TPR Code ... 5-93
5.5.2 Compiling COBOL-85 Code ... 5-94
5.6 Linking a NatStar TPR ... 5-98
5.7 Testing Bull/NatStar Applications ... 5-101
5.7.1 Debugging on the NatStar Development Workstation ... 5-101
5.7.2 Simulating GCOS 7 Native and Conventional Code ... 5-101
5.7.3 Using GCOS 7 Debugging Tools ... 5-102
5.7.4 Debugging in a Simulated Production Environment ... 5-102
5.8 Generating, Compiling, and Linking Client CODE ... 5-103

6. NatStar in a GCOS 7 BATCH Environment

6.1 Preparing a Batch Program 6-1
6.2 Preparing an ORACLE Batch Program 6-7

7. ORACLE Support in TDS/BATCH Mode

7.1 NatStar for GCOS7 with Oracle Support 7-1
7.2 No ORB Support 7-1
7.3 Access Using RPC 7-2
7.4 Restrictions 7-3

8. How to Use the NSCOM Light (NS02OP7G)

8.1 Getting Started 8-1
8.2 The NS02OP7G API 8-3
8.2.1 NSTC_DEFINE_VIRTUAL_NODE 8-3
8.2.2 NSTC_CONNECT 8-5
8.2.3 NSTC_DISCONNECT 8-6

9. How to Use the COBOL Subroutines From the NatStar NCL Language

9.1 Method 9-1
9.2 Examples 9-2

A. Introduction to the NatStar Development Workstation

A.1 NatStar Development Workstation Session A-1
A.2 The Example Application A-3

B. Developing the CONCAT Server Application

B.1 Creating the Server Library B-1
B.2 Defining Server Instructions B-4

C. Developing the Client Window and Logic

C.1 Creating the Local Library C-2
C.2 Defining the Client Window C-3
C.3 Associating the Client and Server Libraries C-11

C.4 Entering NCL for the Client Logic C-13
C.5 Defining the Configuration for the Client Window C-16
C.6 Building Resources for the Concatenate Application C-18
C.7 Testing the Application C-20

D. GCOS 7 Configurations

D.1 Configuring GCOS 7 as a NatStar Server D-1
D.2 Defining the Configuration for COBOL-85 Generation D-4
D.3 Introducing the Connect Disconnect Phase for Remote Servers D-8
D.4 Introducing the UFAS Access File D-11
D.5 Creating a New Server Remote Function D-13

E. Partitioning the Application

E.1 Defining the Remote Servers E-1
E.2 Specifying Remote Operations E-2
E.3 Identifying Instructions as Remote E-4

F. Generating COBOL-85 Source Code

F.1 Generating COBOL-85 Source F-1
F.2 Preparing the Transaction Generation on GCOS 7 ..... F-3
F.2.1 Compile Time F-3
F.2.2 Link Time F-5
F.2.3 TDS Generation F-5
F.2.4 Launching TDS F-5
F.2.5 Launching the NSCOM Daemon and Pseudo Servers on OPEN 7 ..... F-6
F.2.6 Editing the NSCOM.INI File on PC Client F-7
F.2.7 Launching the PC Client Application F-7

G. NCL Sequence Listings

G.1 Enter NCL for the SRVUFAS Library (Server Part) G-1
G.2 Enter NCL for the TESTUFAS Library (Client Part) G-4
G.3 Connection Template G-8

H. NatStar TP Error Codes

I. Error Codes for the Socket Interface on PC

Glossary

Index

Table of Graphics

Figures

1-1. Developing Client/Server Applications Using NatStar 1-3
1-2. Bull/NatStar Client/Server Operation 1-4
2-1. The Client/Server Architecture 2-2
2-2. Operation of Remote Procedure Call 2-3
2-3. Using RPC Stubs 2-4
2-4. Partitioning Client/Server Logic 2-5
2-5. PC Clients and GCOS 7 Server Using NSCOM7 2-8
2-6. Using the Full Bull/NatStar Environment 2-15
2-7. NatStar Code-Generation Options 2-16
4-1. Partitioning Application Logic for Extendibility 4-2
4-2. Lost Update 4-8
4-3. Updating Multiple Databases 4-10
4-4. Step-by-Step Implementation Process with Cross-References 4-14
5-1. Structure and Processing Flow of a NatStar TPR 5-7
5-2. Content of Input Message to TPR 5-8
5-3. Error Popup Window 5-63
5-4. Change Workspace Settings 5-71
5-5. Creating the Language Target Type 5-72
5-6. Creating the User Interface Target Type 5-73
5-7. Installing Services for COBOL Libraries 5-74
5-8. Installing Services for NCL Libraries 5-75
5-9. Main Configuration 5-76
5-10. Target Configuration 5-77
5-11. IM Configuration 5-78
5-12. PM Configuration 5-79
5-13. Main Tab 5-80
5-14. Remote Library Tab 5-81
5-15. Generator Tab 5-82
5-16. Server Configuration Directories 5-83
5-17. Modify Library Description 5-84
5-18. Modify External Representations 5-84
5-19. Modify External Representations 5-86
5-20. Defining a Function or Instruction as Local or Remote 5-87
5-21. Check Remote to Generate Code for a NatStar TPR 5-89
5-22. Generating NCL and Specific COBOL for GCOS 7 5-91

6-1. Modify Function 6-2
6-2. Main Window 6-3
6-3. Targets Window 6-4
6-4. IM Window 6-5
B-1. Display Existing Libraries B-2
B-2. Select the Insert Library Function B-3
B-3. Create the SRVUFAS Library B-3
B-4. Define a New Function B-4
B-5. Prepare to Enter NCL for the Instruction B-5
B-6. Define the Instruction Parameters B-6
B-7. Enter NCL for the SVR_CONC Instruction B-7
C-1. Create the TESTUFAS Library C-2
C-2. Define the Client Window, Part 1 ..... C-3
C-3. Define the Client Window, Part 2 ..... C-4
C-4. Prepare to Define the User Interface C-5
C-5. Define the Display Text for the First Operand C-6
C-6. Define the Entry Field for the First Operand C-7
C-7. Define the Entry Field Type for the First Operand C-8
C-8. Define the Concat Push-button C-9
C-9. Complete the User Interface Window C-10
C-10. Access the TESTUFAS Library Description C-11
C-11. Give TESTUFAS Library Access to SRVUFAS Library C-12
C-12. Select the Executed Event for WINUFAS C-13
C-13. Enter NCL for the Concat Push-button C-14
C-14. Create the WINUFAS Window Configuration, Part 1 ..... C-16
C-15. Create the WINUFAS Window Configuration, Part 2 ..... C-17
C-16. Build Resources for the Concatenate Application C-19
C-17. User Interface for Concatenate Application C-20
D-1. Creating the Language Target Type D-2
D-2. Creating the User Interface Target Type D-3
D-3. Define Configuration for COBOL-85 Generation D-4
D-4. Define COBOL-85 Generation as a Server Configuration D-5
D-5. Link COBOL-85 Generation Configuration to the User Interface Definition D-6
D-6. Define Pathnames for COBOL-85 Source D-7
D-7. Importing the G7USRPASS library D-8
D-8. Defining the Connection Template D-10
D-9. NCL Code in EXECUTED Event of PB_LISTE Push Button D-12
D-10. SRV_GET_UFAS Function, Parameters Definition D-13
D-11. Displaying the Code Executed on the Server Side for the UFAS Access D-14
E-1. Enter NCL for WINUFAS INIT Event E-2
E-2. Specify Remote Operations for the WINUFAS Calculate Application E-3
E-3. Define Instruction as Remote E-4
E-4. Rebuild the Client Configuration E-6
F-1. Select SRVUFAS Library for NCL Generation F-2
F-2. Modify Library Description F-3
F-3. Modify External Representations F-4
F-4. Operational Connected Screen Aspect of the Application TESTUFAS F-8

Tables

2-1. NATSTAR/TPRs and Conventional TPRs 2-14
5-1. NCL Data Types that Map to COBOL Data Types 5-47
5-2. Converting COBOL Data Types Not Supported in NCL 5-49
5-3. Status Values Returned by the NCL Data-Conversion Routines 5-52
5-4. Data-Type Codes Accepted by NCL Data-Conversion Routines 5-53
5-5. Mapping NCL Variable Names to Names Used in Native Code..... 5-57

1. Overview

Bull/NatStar is one of the Bull product offerings that integrate GCOS 7 systems into the open systems environment. Bull/NatStar supports a client/server environment in which GCOS 7 systems act as servers for client applications that run on PCs or UNIX workstations. Bull/NatStar is a joint development of Bull and Nat Systems. Bull/NatStar extends the application-development toolset supplied by Nat Systems to include support for server applications that run under TDS on GCOS 7.

Bull/NatStar simplifies the implementation of client/server applications through the use of NatStar, the Nat Systems development environment. NatStar consists of visually oriented second-generation application-development tools that can operate on either a PC or a UNIX workstation.

First-generation application-development toolsets focused on the client side of client/server applications, and assumed that the server side would be developed using conventional methods. In contrast, NatStar allows you to develop *both* the GUI-based client side *and* the associated GCOS 7-based server side of the client/server application at the same time, using the same tools and methods. Another benefit of NatStar is that it automatically generates the software required to manage interactions between the client and the server.

This chapter provides a general overview of the contents of this guide, and defines some terms that are used throughout the document. The following topics are included in this chapter:

- Purpose of the document
- Features and benefits of the Bull/NatStar environment
- A guide to the use of this document

1.1 Purpose of this Document

This document explains how to develop client/server applications to operate in the Bull/NatStar environment.

Most of the information about how to develop and debug client/server applications using NatStar is not specific to GCOS 7. As a result, that information is provided in the documentation available from Nat Systems (the order numbers of NatStar documents are provided in *Bibliography* in the *Preface* of this Guide).

This document focuses on the aspects of client/server development that are unique to the use of GCOS 7 as a server system. These aspects include:

- Using NatStar to generate COBOL-85 code to be executed on GCOS 7
- Transferring the generated COBOL-85 code to GCOS 7, then compiling and linking it for use with TDS
- Configuring the TDS environment(s)

The information contained in this document is intended for use by developers of client applications and TDS server applications, and by GCOS 7 System Administrators.

1.2 Features and Benefits of Bull/NatStar

Bull/NatStar combines the second-generation NatStar application-development environment with GCOS 7 support for client/server applications developed using this environment.

NatStar is a visually oriented, full life-cycle toolset for building enterprise applications that use the client/server architecture. A key benefit of NatStar is that you can develop a client/server application and debug it interactively without any need to specify where specific logic will be executed. Then by simply defining one or more segments of logic as remote, you can distribute those segments and NatStar will provide the necessary support for interaction between the client and server portions of the application.

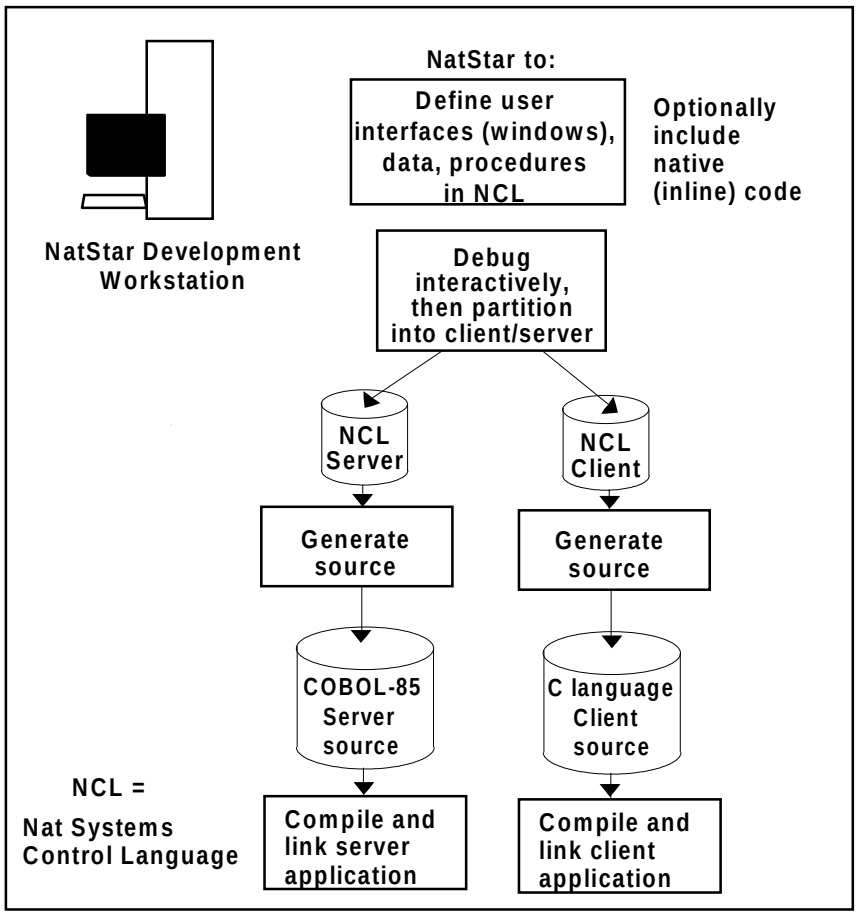

Figure 1-1. Developing Client/Server Applications Using NatStar

Client/server applications generated using NatStar are platform- and network-independent. You can define the client/server logic using NCL (Nat Systems Control Language), which is the Nat Systems development language.

After you have partitioned the debugged NCL into client and server parts, server source code is generated in COBOL-85. The code is adapted to the target environment only when COBOL-85 source is compiled and linked on the target platform.

NOTE:

NCL provides language primitives that can be used to access a GCOS 7 UFAS file. Also, programs coded in NCL can access GCOS 7 databases and files indirectly by calling a conventionally coded user subroutine that accesses the database or file.

A complete description of the facilities of NatStar is outside of the scope of this document. Refer to the Nat Systems documentation, order numbers for which are listed in *Bibliography* in the *Preface* of this manual. This document focuses on the components of NatStar that are related to its use with GCOS 7.

The major Bull/NatStar components involved in the operation of a client/server application are shown in Figure 1-2.

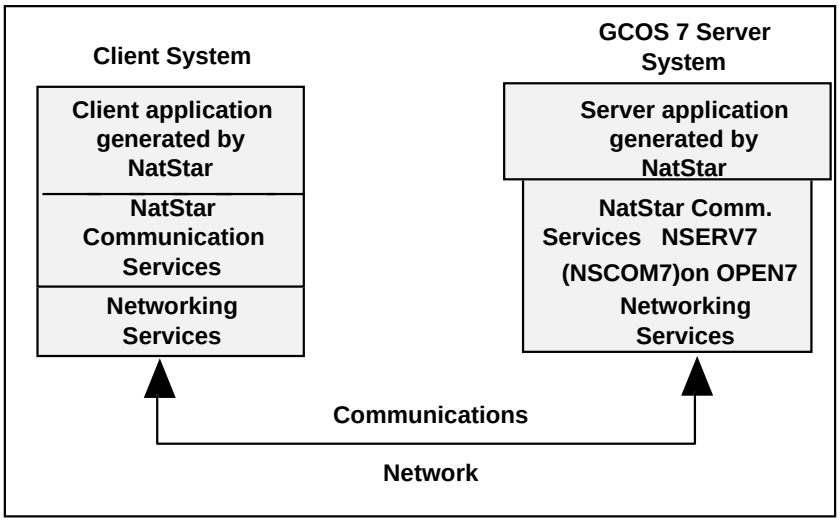

Figure 1-2. Bull/NatStar Client/Server Operation

The components shown in Figure 1-2 are as follows.

- The client and server applications are generated by NatStar, then compiled and linked into the operating environment.
- The NatStar Communication Services software organizes and manages the interaction between the client and server applications. NatStar Communication Services include the runtime routines used during execution of the client/server application, and a special server for each TDS associated (NSERV7). NSCOM7 is the GCOS 7 implementation of the NatStar Communication Services.
- The Networking Services provide the network interfaces and Transport Services necessary to transmit requests, responses, and data between the client and server applications.

Benefits of the Bull/NatStar Environment

Bull/NatStar offers the following benefits.

- **Improves designer productivity by simplifying the implementation of client and server code.** NatStar is easy to use, and enables the simultaneous development of both the client and server sides of an application. NatStar combines the simplicity of a visually oriented development environment with the power of an easy-to-use development language, NCL.
- **Minimizes problems in client/server interaction.** No hand-coding is required to manage the interaction between client and server applications. NatStar automatically generates the code for the Remote Procedure Call (RPC) protocol, which the client and server use to exchange requests and responses.
- **Offers flexibility in client/server partitioning.** After you have completed the development and logical debugging of the client/server application, you can direct NatStar to distribute specific logic to one or more server systems (all other logic remains in the client). You can later re-partition the application to reassign logic as needed because of hardware reconfigurations and/or to improve performance.
- **Is compatible with the GCOS 7 environment.** Code generated by NatStar can operate in the TDS environment in the same way that hand-coded TPRs operate under TDS. NatStar-generated code can be combined with conventionally developed code, and can also be used outside of TDS, for example as a subroutine in a batch program.
- **Supports multiple client/server configurations.** The Bull/NatStar environment supports a variety of client/server configurations, including those that involve two, three, or more types of client and server platforms. These configuration options are described in Section 2.2 *Bull/NatStar Support of the Client/Server Architecture*.

1.3 A Guide to the Use of this Document

You can use this document for various purposes, and in various ways. Here are some suggestions for how to most effectively use this guide.

If you want a quick start in learning about Bull/NatStar . . .

Load the NatStar software on your PC, as described in the Nat Systems documentation and/or in the *Bull/NatStar Software Release Bulletin*.

Then go to Appendix A, *Introduction to the NatStar Development Workstation*, and work your way through the example to develop a simple client/server application that runs on the PC and GCOS 7.

If you are not familiar with the NatStar software, first go to the *NatStar Getting Started* manual (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface*) and work your way through the example in which you develop a generic client/server application.

Then go to Appendix A, *Introduction to the NatStar Development Workstation*, and do the GCOS 7-specific client/server application.

If you are new to client/server environments . . .

If this is the first time you are developing or administering client/server applications, first read Chapter 2 *Bull/NatStar in the Client/Server Environment* for some basic concepts and to see how Bull/NatStar supports those concepts.

Then read Chapter 4 *Designing Client/Server Applications* for some ideas about the high-level design of client/server applications. When you understand the concepts in these two chapters, you are ready for Chapter 5 *Implementing Client/Server Applications*, which provides specific information about how to develop GCOS 7-based client/server applications.

If you want to know what is different in developing TPRs for Bull/NatStar . . .

If you are an experienced developer of TPRs (to operate under TDS) and you want to know what factors you need to consider in developing TPRs, then read Chapter 5 *Implementing Client/Server Applications* which describes the structure of NSCOM7 TPRs and summarizes the design and implementation factors that are different in this environment as compared with a host/terminal TDS environment.

Pay particular attention to the discussion in Section 5.5 *Special Considerations When Developing an NSCOM7 TPR* which focuses on how developing an NSCOM7 TPR differs from development of a conventional TPR.

Read also Chapter 4 *Designing Client/Server Applications* to be sure that you are aware of the high-level design concepts that are different from a host/terminal TDS environment.

2. Bull/NatStar in the Client/Server Environment

This section provides an overview of how Bull/NatStar can be used in a client/server environment. Understanding the features and options associated with Bull/NatStar is an important prerequisite for understanding how to implement client/server applications for this environment.

The following topics are discussed:

- The client/server architecture, which introduces basic concepts used throughout this document.
- Bull/NatStar support of the client/server architecture, which illustrates the types of client/server configurations supported.
- Options for using Bull/NatStar, which describes various ways in which Bull/NatStar capabilities can be applied.

2.1 The Client/Server Architecture

A general understanding of the client/server model is useful to anyone who will be working with Bull/NatStar.

NOTE:
If you are already familiar with the client/server architecture and the use of Remote Procedure Call (RPC), you may wish to skip this section.

2.1.1 Basic Client/Server Concepts

Client/server has emerged as the dominant technique for the development and deployment of distributed applications. In the client/server architecture. Some applications or processes act as **clients**, while others act as **servers**. A general model of the client/server architecture is shown in Figure 2-1.

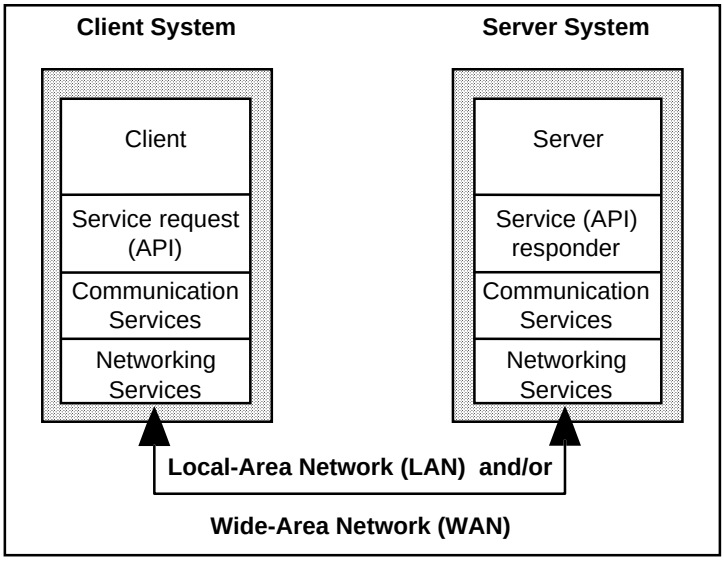

Figure 2-1. The Client/Server Architecture

A client/server application is a distributed application in which:

- The client sends a request for service (typically by using an application programming interface, or API) to the server.
- In response to the request, the server provides services.
- The server sends information back to the client.

2.1.2 Using Remote Procedure Call (RPC)

There are various ways for the client and server portions of the distributed application to communicate. The method used in Bull/NatStar is the Remote Procedure Call (RPC). The use of RPC is illustrated, in overview form (only the most important software elements are shown), in Figure 2-2.

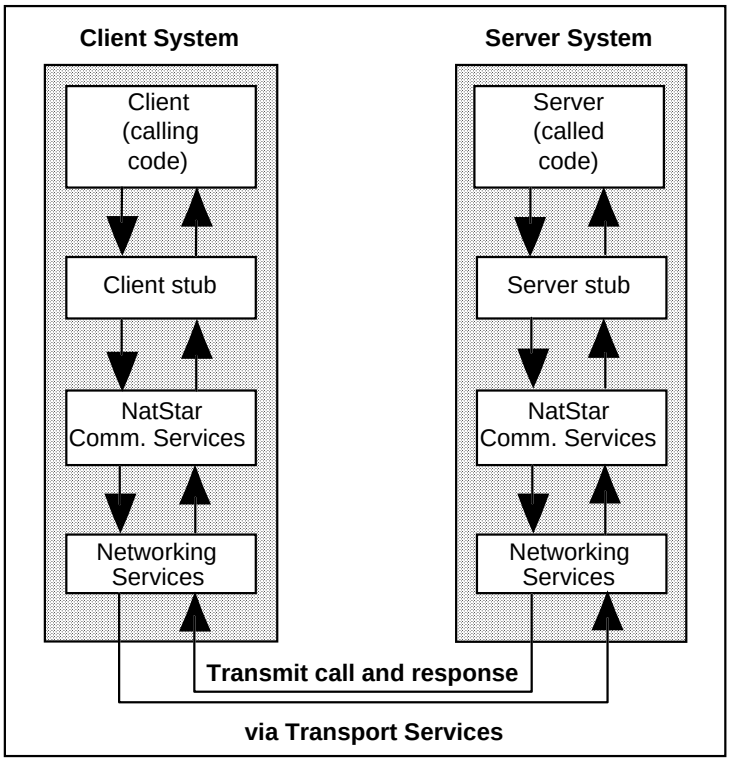

Figure 2-2. Operation of Remote Procedure Call

Using RPC makes the distribution of application logic transparent to the client and server applications. Transparency allows the client and/or server to be moved without changing the logic of either application. The specific network protocol being used is also transparent to the client and server.

This transparency is achieved by creating **stubs** at the client (calling) location and at the server (called) location. Stubs consist of software that coordinates the exchange of calls and responses, and that manages data conversions when required. In Bull/NatStar, the code for the stubs is generated automatically.

A major advantage of RPC is that developers do not have to learn the details of communication APIs.

The use of RPC stubs is illustrated in more detail in Figure 2-3.

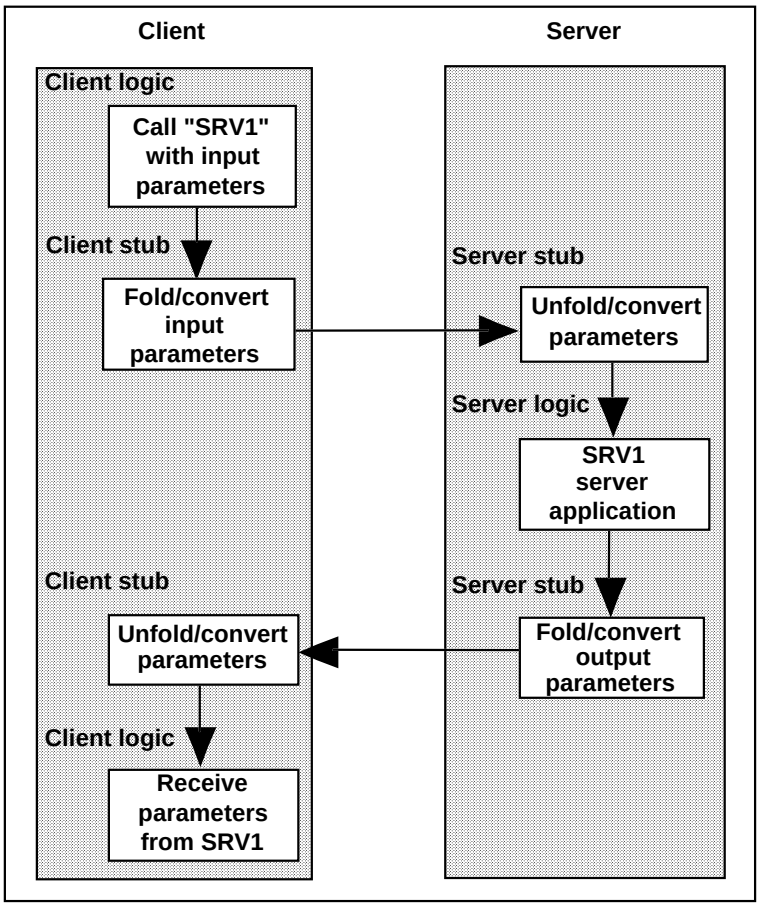

Figure 2-3. Using RPC Stubs

The main purpose of the client and server stubs is to carry out whatever conversions are necessary to make the data understandable to the receiver (whether server or client). For example, when transferring parameters and data between a client on a PC, which uses a 32-bit data ASCII format, and a server on GCOS 7, which uses a 32-bit data EBCDIC format, the data must be transformed so that it can be processed by the receiver.

2.1.3 Client/Server Configurations

The client/server architecture can be used to create configurations that range from simple to very complex. This section discusses some general concepts associated with client/server configurations. The configurations supported by Bull/NatStar are described later in this chapter.

Partitioning Client/Server Logic

The "classical" way to partition application logic across a client/server configuration is to use the architecture shown in Figure 2-4.

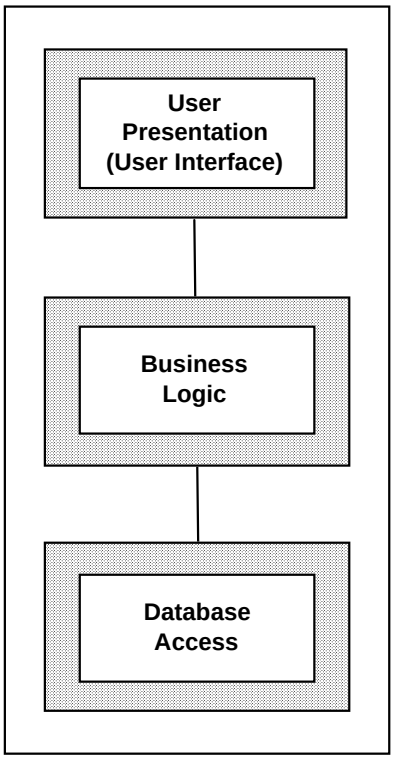

Figure 2-4. Partitioning Client/Server Logic

The architecture shown in Figure 2-4 partitions the client/server application logic into three parts:

- the logic that is directly concerned with interaction with the user (User Presentation).
- the processing logic of the application (Business Logic), based on the requirements of the business or organization that uses the application.
- the logic that accesses databases and/or files.

A client/server application designed using this architecture can be partitioned cleanly across whatever number of platforms is appropriate in the specific situation.

An approach sometimes called **two-tier client/server** assigns the presentation and business logic to the client, and assigns only database access to the server. This results in a **"fat client"** because most of the application logic is included in the client.

An approach sometimes called **three-tier client/server** assigns only the presentation logic to the client and assigns the business logic and database access to one or more servers. This results in a **"thin client"** (or **"skinny client"**) because only the user interface is included in the client.

As long as the logical partitions shown in Figure 2-4 are maintained, the assignment of logic to platforms can be modified as necessary. For example, both the business logic and database access could be assigned initially to the same server. Later, the business logic could be moved to a different server without changing the logical relationships of the client/server application, and without causing any change to the user presentation.

IMPORTANT:

The computer industry does not as yet use completely consistent terminology to describe client/server configurations. In some cases, terms such as "two-tier" and "three-tier" are used to indicate logical partitioning, without regard to how many different platforms are included in the configuration. In other cases, these terms are used to indicate the number of different types of platforms (e.g., a client PC and a UNIX server making a two-tier configuration). Because of this inconsistency, the terms "two-tier", "three-tier", etc. are not used to describe the capabilities of Bull/NatStar.

2.2 Bull/NatStar Support of the Client/Server Architecture

The Nat Systems *NatStar Development Guide -- Client-Server* lists the ways in which application logic can be distributed transparently (the order numbers of Nat Systems documents are provided in the preface of this Guide). Using the RPC mechanism, which is the method supported in Bull/NatStar, the *NatStar Development Guide -- Client-Server* states that you can distribute:

- Functions
- Instructions
- Database access

In NCL, a **function** and an **instruction** are both ways to define a block of coding logic. A function returns a value that is usually a status return, but could be a result value. An instruction does not return a value. A function and an instruction can each have both input and output parameters.

IMPORTANT:
Functions and instructions are concepts that are important in NatStar. When you distribute code generated by NatStar to a GCOS 7 system, functions and instructions both become blocks of GCOS 7 code, which you then use to create GCOS 7 transactions.

NOTE:
It is possible for an NCL function or instruction to call a GCOS 7 subroutine.

UFAS file access on GCOS 7 is generated using the NatStar development environment.

This section consists of illustrations and explanations of the most important client/server configurations supported by Bull/NatStar.

2.2.1 Using Bull/NatStar To Implement Client/Server Applications

Bull/NatStar supports several configurations that include two types of platforms -- one type of client system and one type of server system -- interacting in client/server mode. This section provides some examples of applications that use such a configuration.

2.2.1.1 PC Clients and GCOS 7 Server Using NSCOM7

Figure 2-5 illustrates a client/server application in which clients on Windows PCs interact with servers on GCOS 7.

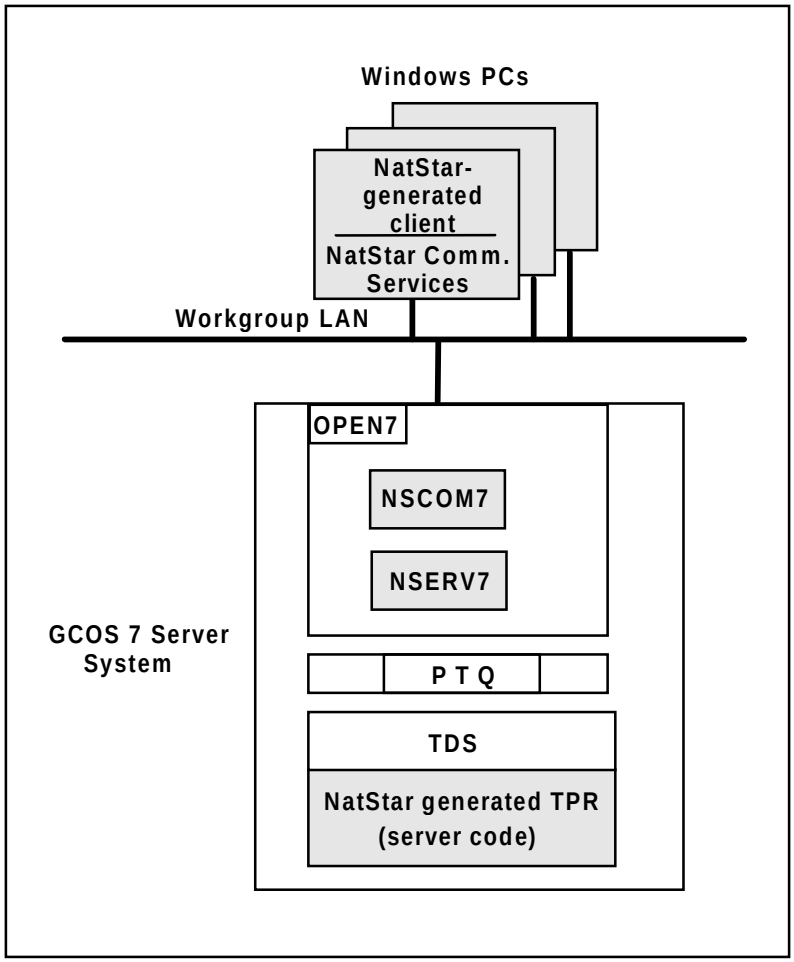

Figure 2-5. PC Clients and GCOS 7 Server Using NSCOM7

In this and later figures, the code segments associated with Bull/NatStar are identified by being shaded. Client and server applications are associated with the NatStar Communication Services (NSCOM7, and NSERV7 on OPEN 7/GCOS 7).

A client/server configuration such as the one shown in Figure 2-5 has the following characteristics.

The client systems are attached via a LAN to OPEN 7 running on a DPS 7000.

The network protocol stack required for the exchange of data between the clients and GCOS 7 is installed on OPEN 7. Typically software to support TCP/IP connections from clients to GCOS 7 is also installed on the OPEN 7.

A platform-specific version of NatStar Communication Services is installed on each of the client and server systems; GCOS 7 uses NSCOM7. NSCOM7 runs on OPEN 7 and is the full NSCOM implementation. For each TDS running on the DPS 7000, an NSERV7 server runs on OPEN 7 and waits for client requests. An NSERV7 server manages all the connections for a given TDS: it submits client requests as transaction processes under TDS.

2.2.1.2 Launching the NSCOM Daemon on OPEN 7

For more information on launching the NSCOM daemon on OPEN 7, refer to the NatStar documentation (NatStar/Communication Services).

All the executable files are in the directory **/usr/natsys/bin**, and the NSCOM.SYS file is in the directory **/usr/natsys/ini**.

Usually the NSCOM daemon is launched automatically when OPEN 7 is started.

The procedure for starting the NSCOM daemon manually when necessary is as follows:

Log in as root user, then enter:

```
cd /usr/natsys/ini
. ./inivar              set the OPEN 7 environment variables
cd ..
nsserver   -start       launch the NSCOM daemon
```

To stop the NSCOM daemon `nsserver` once the environment variables are set:

```
nsserver  -stop
```

2.2.1.3 Launching and Administering NSERV7

NSERV7 is an NSCOM server which is dedicated to connecting the PC client with a given TDS on GCOS 7.

This module is delivered on the OPEN 7 partition **usr/natsys/bin/nserv7**, with the NatStar/GCOS 7 integration product.

In this architecture there is one NSERV7 server per TDS, and the launching parameters for NSERV7 are the following:

-t	Mandatory parameter, name of the TDS running on GCOS 7.
-d	Optional parameter, if present the NSERV7 server will trace all the messages between the PC client and TDS.
-c	Optional parameter, if present defines the maximum number of authorized connections for this server. By default, the maximum number of connections is 200. The theoretical maximum value of this parameter is 255.

Since NSERV7 specifies the number of connections, it must be accompanied by configuration of the OPEN 7 pool of semaphores.

The maximum number of semaphores and the size of the semaphore set must be configured as follows:

1. Total number of semaphores

   Each TDS application uses three lock semaphores and one global lock semaphore, so the total number of semaphores is:

   maximum number of connections for all TDSs + 3 x number of services + 1

2. Maximum number of semaphores in a semaphore set.

   The maximum number of semaphores in a semaphore set is the maximum number of connections for any one NatStar TDS + 3

EXAMPLE:

If NatStar generates two TDSs, one supporting 500 connections, the other supporting 200 connections, add the following two lines in the `config_user.h` file (in **/makopen7/cf**):

```
#define SEMMSL          503 /* max sem per sem set */
#define SEMMNS          707 /* total of sem        */
```

❑

You must also take into account the semaphores used by other applications running under OPEN 7. See the *OPEN 7 Administrator's Guide* (paragraph *Configuration of OPEN 7*).

The environment variable NSCOM= (path of the **nscom.ini** file) must be present. It is declared in the file `/usr/natsys/ini/inivar` which contains:

```
#cat inivar
NSCOM=/usr/natsys/ini/nscom.ini
export NSCOM
PATH=$PATH:/usr/natsys.bin:./
export PATH
NSOPEN7=/usr/natsys/ini
export NSOPEN7
#
```

This server must be configured on OPEN 7 with the **nscom.ini** initialization file.

Manual Launching

The **nscom.ini** file on OPEN 7 must contain the following block for the SRVSERVICES structure for each server to be run:

```
Service =SCHA
       CplService       =
       ProcessId        = /usr/natsys/bin/nserv7 -t scha -d
       Start            =
       SrvTimeout       = -1
       SrvAValidDelay   = -1
       WorkDirectory    =
       Trace            = NO
       MinSessLimits    = 2
       MaxSessLimits    = 32
```

where:

SCHA	is the name of the service,
scha	is the name of the TDS on GCOS 7,
start	is empty
MinSessLimits	is set to 2 to launch the NSERV7 process twice manually (`nsserver` must be launched before NSERV7 as described above).

Execute the following OPEN 7 commands as root user:

```
nserv7  -t 'TDS-NAME'  [ -d]
nserv7  -t 'TDS-NAME'  [ -d]
```

To stop the NSERV7 server, use the UNIX command **kill -9** , or stop the NSCOM daemon `nsserver`.

Automatic Launching

For automatic launching, the NSCOM.INI file must contain the following block in the SRVSERVICES structure, with the parameter START set to AUTO.

```
Service =SCHA
       CplService       =
       ProcessId        = /usr/natsys/bin/nserv7 -t scha -d
       Start            = AUTO
       SrvTimeout       = -1
       SrvAValidDelay   = -1
       WorkDirectory    =
       Trace            = NO
       MinSessLimits    = 2
       MaxSessLimits    = 32
```

With this configuration, NSERV7 is launched at the same time as the NSCOM daemon `nsserver`.

To stop the NSERV7 server, you must stop the NSCOM daemon `nsserver` because, if you use the UNIX " kill -9 " command, `nsserver` will automatically restart NSERV7 immediately.

Dynamic Launching

For dynamic launching, the NSCOM.INI file must contain the following block in the SRVSERVICES structure, with the parameter START set to DYNAMIC.

```
Service =SCHA
       CplService      =
       ProcessId       = /usr/natsys/bin/nserv7 -t scha -d
       Start           = DYNAMIC
       SrvTimeout      = -1
       SrvAValidDelay  = -1
       WorkDirectory   =
       Trace           = NO
       MinSessLimits   = 2
       MaxSessLimits   = 32
```

The NSERV7 server will be launched after the NSCOM daemon, on the first request of the PC client.

To stop NSERV7, you can either use the UNIX " kill -9 " command, or stop the NSCOM daemon `nsserver`, but NSERV7 will restart at the first client request.

Administration of NSERV7

These three commands (run as root in an NSCOM environment) allow you to administer the NSERV7 servers:

`nserv7adm` lists all TDS services running. For each service, the number of running NSERV7 servers and the number of users connected to the TDS are listed.

`nserv7adm -t <TDS name >`
lists all users connected to the TDS.

`nserv7adm -t <TDS name > -d <user name>`
allows you to disconnect a user from a TDS.

LOG FILE of errors messages from TDS

If a fatal error appears when running a TPR (ABORT TPR, etc.), NSERV7 reads a VCAM error message in the mailbox and cannot transmit the corresponding NSCOM frame. It then sends an error message to the NSCOM client giving the class of the problem and automatically puts the error message read in the mailbox into the `<TDSNAME>.log` file (the name of the TDS is in uppercase). The log file is always written to in append mode and each message is preceded by the date and time.

This file is created when the first error message is sent to the directory specified by the NSOPEN7 environment variable. The messages concerned are those sent during a TDS session; they are identified by the NAT 5xx and are listed under the heading *Specific TDS Errors Collected by NatStar* in Chapter 5 *Implementing Client/Server Applications*.

2.3 Options for Using Bull/NatStar

Using the full Bull/NatStar environment typically provides the most advantages, but you can also use subsets of the environment as described in this section.

2.3.1 NatStar TPRs and Conventional TPRs

The relationships shown in Table 2-1 are important in understanding the Bull/NatStar options.

Table 2-1. NATSTAR/TPRs and Conventional TPRs

	Uses NatStar Communication Services	**Does not use NatStar Communication Services**
Developed using NatStar	***NATSTAR TPR***	***Batch Mode***
Developed using conventional methods	***NOT APPLICABLE***	***Conventional TPR***

The matrix in Table 2-1 explains the meanings of the following terms:

NatStar TPR — A NatStar TPR always uses the NatStar Communication Services. A NatStar TPR is developed using the NatStar development workstation.

Conventional TPR — A conventional TPR or subroutine does not use NSCOM. A conventional TPR typically is developed using conventional methods (e.g., by programming in COBOL-85). In this version a conventional TPR cannot be called from a NatStar TPR.

2.3.2 Using the Full Bull/NatStar Environment

The full Bull/NatStar environment provides development and execution support for client/server applications. Assuming that you want to develop a new client/server application to operate on PCs and GCOS 7, the components highlighted in Figure 2-6 are involved.

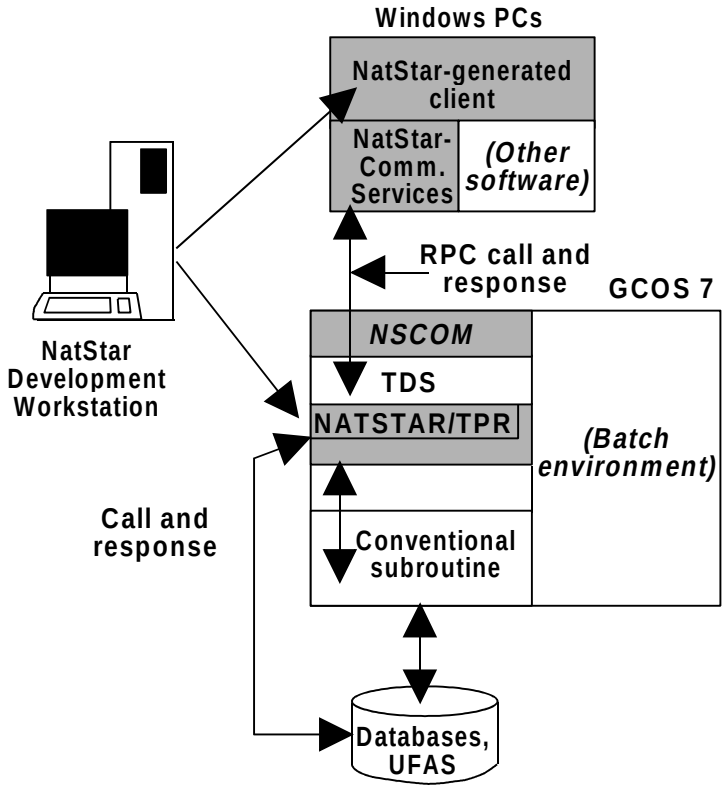

Figure 2-6. Using the Full Bull/NatStar Environment

In the full Bull/NatStar environment, NatStar is used to generate client code and server code. After compiling and linking on the target platforms, the client and server code communicates using RPC via the NatStar Communication Services on GCOS 7/OPEN 7.

You will need to use conventional code to access databases. The NatStar TPR can call an existing subroutine for this purpose.

2.3.3 Using NatStar Independently

You can use NatStar independently, to generate code that does not use the NatStar Communication Services. This code can be compiled and linked on GCOS 7. However, the focus in this document is on GCOS 7.

Code generated by NatStar conforms to ANSI COBOL-85 standards. You can use the generated code in GCOS 7 applications other than client/server applications, or in client/server applications that do not use NSCOM. For example, NatStar-generated code can be used as:

- a subroutine to be called in batch
- a subroutine to be called by a TPR
- in any other situation where COBOL-85 source code is used.

Figure 2-7 illustrates this option in simplified form.

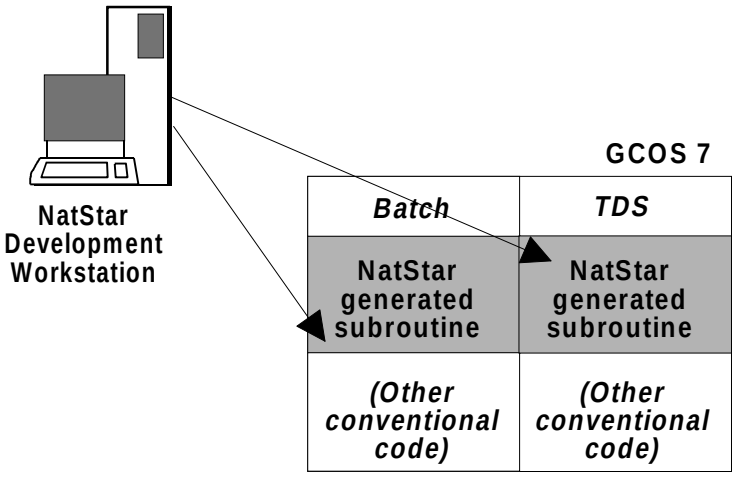

Figure 2-7. NatStar Code-Generation Options

NOTE:
Although the shaded code is generated by NatStar, it is conventional code because it does not use NSCOM. You must link this code with PSL7 Compile Unit library for referencing the NCL runtime routines ("SI7.NS7.CULIB").

3. Planning for the Bull/NatStar Environment

This section discusses the topics you need to consider in planning for the installation of the Bull/NatStar environment. Typically this planning process is the responsibility of a System Administrator.

The following topics are included in Chapter 3:

- Planning for required software and hardware.
  The specific hardware and software components supported by NatStar are described in the *NatStar Installation and User's Manual*.
- Configuring the TDS environment.

3.1 Planning for Required Software and Hardware

To use Bull/NatStar, the following components are required:

1. GCOS 7 operating system (plus TDS), which will execute the server applications (NatStar TPRs) that you develop.
2. OPEN 7 subsystem which will run the NSCOM7 or OPCN7 Gateway.
3. NatStar development workstations, which you will use to develop your client/server application logic, and to execute the client application.
4. Optionally, NatStar workstations that execute client applications but without having any application development capability.

The following subsections specify the required and optional software for each of the above components. All components that are *not* labeled "delivered with NSCOM" are prerequisites that must either be:

- already installed at the site, or
- ordered separately and installed before NatStar Communication Services are installed.

3.1.1 GCOS 7 Required and Optional Software and Hardware

The following list specifies the required and optional software and hardware when GCOS 7 is used with Bull/NatStar.

1. GCOS 7 operating system (at least V7 TS 7356).
2. OPEN 7 subsystem (at least V4, or V5 if you use NSCOM Light).
3. TDS release.
4. COBOL-85, to code, generate and compile NatStar TPRs in COBOL-85.
5. TCP/IP software on OPEN 7 with FCP7 or ISL.
6. File transfer software.
7. NSCOM7 (NSCOM7,NSERV7,PSL7) V 1.0 or NSCOM Light, automatically installed by ISI7.

3.1.2 NatStar Development Workstation Required Software

The following list specifies the required software when NatStar development workstations are used with Bull/NatStar.

1. Supported operating system.
2. NatStar current release, with any accompanying Service Pack.

See the *GCOS 7/NatStar Integration Software Release Bulletin* for full details of the software required.

NOTE:
Client applications that execute on NatStar development workstations or client systems use TCP/IP to interface with server applications on GCOS 7 via OPEN 7/FCP7 or ISL.

3.2 Configuring TDS

No configuration is required on the GCOS 7 side to use a NatStar TPR (it is only necessary to integrate a new TPR in a given TDS in the usual way).

4. Designing Client/Server Applications

This section discusses topics to be considered during the high-level design phase for Bull/NatStar applications. These topics apply to applications which are discussed in Chapter 5 *Implementing Client/Server Applications*.

The following topics are included in this chapter:

- Ensuring extendibility, which discusses how to structure client/server applications that can easily be changed or reconfigured.
- Achieving satisfactory performance.
- Maintaining data integrity.
- Implementing security controls, which includes a discussion of differences in security-control techniques between a traditional GCOS 7 environment and a GCOS 7-based client/server environment.
- A step-by-step implementation guide for developing Bull/NatStar client/server applications.

4.1 Ensuring Extendibility

The key to successful extendibility and the reuse of logic is the appropriate structuring of applications and the definition of stable interfaces to insulate each layer from changes in other layers. The classical model is that of partitioning an application into three layers, as shown in Figure 4-1.

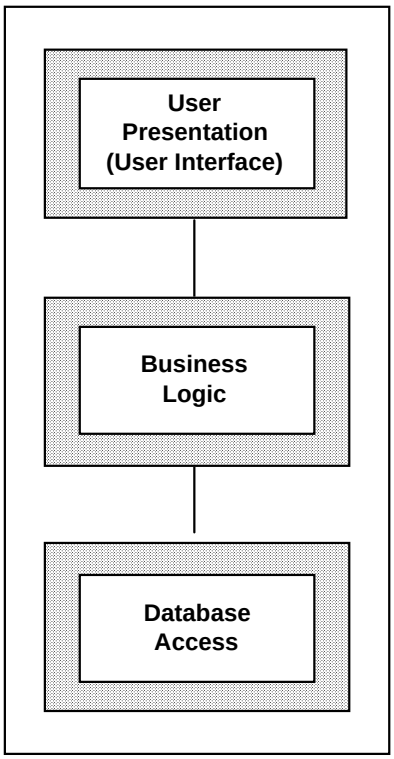

Figure 4-1. Partitioning Application Logic for Extendibility

With a properly structured application, any one layer can be changed without impacting the others, thus allowing reuse. For example, the presentation and business-logic layers could be reused even if the underlying physical database is restructured or converted from IDS/II to relational. Similarly, the business-logic and database-access layers could be reused if the presentation layer is changed from terminal forms to an RPC mechanism.

A Bull/NatStar application typically has the presentation layer on the desktop, the business-logic layer on a server, and the database-access layer on the same or a different server. A TPR in this environment typically is composed of a business-logic layer generated by NatStar, which calls a database-access layer that has been generated conventionally (not using NatStar). This method enables access to existing GCOS 7 databases and the potential reuse of existing application code for database access.

The following three principles are ***very important*** in achieving the goal of extendibility:

1. Isolate database access from business logic by using database-wrapper routines. The business logic should never call the database-access routines directly, but instead should always use the application wrappers. This method ensures a stable interface between business logic and database access.
2. Minimize the amount of business logic included in the client. If business logic is included in the client, the software in every client system must be updated each time that business logic changes. Propagating changes to dozens or hundreds of client systems is a challenging task. You should keep this fact in mind during the design of each client/server application.
3. Maintain stable interfaces between user presentation and business logic. If these interfaces are designed so that frequent changes are required, then you will have to update the client code on all client systems each time a change is made.

4.2 Achieving Satisfactory Performance

Achieving satisfactory performance is typically the greatest challenge in designing client/server applications. From the user's perspective, response time is the key performance criterion. From the overall system perspective, throughput is an additional concern.

Important hardware considerations in achieving satisfactory performance include the power of the client systems (CPU, memory), the power of the server system(s) (CPU, memory, disk performance), and the speed and complexity of the network (type and speed of network facilities, number of links, protocol mappings between client and server, etc.). However, the design of the application itself is generally a limiting factor in determining the achievable performance.

As in conventional programming, database design and issues related to concurrent access typically have the greatest impact on performance.

A complete discussion of techniques for analyzing and optimizing performance is beyond the scope of this document, but you need to consider the following key factors.

4.2.1 Minimizing the Number of Client/Server Interactions

Minimizing the number of interactions between the client and server is essential to achieving satisfactory performance. Methods for minimizing interactions include:

- Structure the client to collect all of the required data from the user, validate the data, then pass all of the data to the server. Do not pass data elements one-by-one to the server.
- Structure the client to retrieve all of the required data from the server in a single request. Do not make multiple requests to the server to retrieve multiple data rows or records.
- Consider caching read-only data on clients or on a workgroup/network server to minimize client requests to the server. For example, in an order-entry application, shipment and payment terms and conditions for all orders could be cached.

IMPORTANT:

If you plan to cache data on clients or on workgroup or network servers, be sure to analyze the situation carefully. If you cache data that can change during production processing, you will need to implement an algorithm by which the client can validate the contents of cache and update values if necessary. In general, data that cannot change during production processing is the best choice for caching.

4.2.2 Minimizing the Volume of Data Transferred

In addition to minimizing the number of client/server interactions, the design should minimize the volume of data transferred, thus minimizing network traffic and the associated overhead.

Data-intensive activities should be performed on the server if possible. For example, use the server to perform an operation such as computing the average of some value from a set a records, rather than moving those records to the client and computing the average there. This method improves performance by reducing network traffic, and also takes advantage of the data-manipulation and processing capabilities of the server system.

The object-oriented design view supported by the NatStar application-development tools can help you to model the client/server interface in terms of requests for functions that operate on data. Although the Bull/NatStar implementation is not object-oriented, you can use OO design methods to ensure that logic is partitioned to minimize interactions and data transfers between the client and server.

4.2.3 Optimizing Network Performance

The performance of the network that connects the clients with the server(s) has an important impact on application performance. Consider the following factors:

- Client/server performance over a LAN typically is superior to performance over a WAN, for two reasons: 1) a LAN typically provides higher bandwidth; and 2) LAN protocols involve less overhead than WAN protocols such as X.25.
- Minimizing the number of interactions and the quantity of data transferred is especially important when a WAN is used.
- Consider caching data on the clients or on workgroup or network servers. (Refer to the note about caching data in Section 4.2.1 *Minimizing the Number of Client/Server Interactions*.)

The Bull/NatStar implementation includes some built-in features that can help to optimize network performance.

- The NatStar Communication Services software in the client will reuse an existing session with a server for subsequent interactions, thus avoiding the overhead associated with establishing a new session.
- NatStar Communication Services (including NSCOM7) use a computer-to-computer protocol rather than a terminal-oriented protocol, which reduces the number of interactions across the network by approximately 50% because of the elimination of software acknowledgments.
- Only data is sent from GCOS 7 servers to clients; no format information is required. In many applications the result is that less data is sent from servers to clients.

4.3 Maintaining Data Integrity

All of the facilities of TDS and GCOS 7 for managing data integrity are available to the designer of NSCOM7 server applications. Those facilities provide a great deal of flexibility in managing the reservation and release of resources during dialog with the client application. However, special care is required to maintain data integrity in a client/server configuration.

IMPORTANT:
Each RPC call/response between a client and a NatStar TPR forms a commitment unit. This is the default option in transaction generation under TDS.

Because GCOS 7 Integrity Control does not implement the X/Open Distributed Transaction Processing (DTP) model, each RPC call/response -- i.e., each interaction between the client and server -- forms a commitment unit on GCOS 7. Establishing a commitment unit for every interaction imposes some constraints on the overall design of client/server applications. Those constraints must be taken into account to maintain database integrity while producing the response expected by the users.

If each commitment unit is not structured properly, the following problems can occur:

- Updates to GCOS 7 databases may not be consistent with updates to other databases -- i.e., on the client system or on a UNIX server -- that are associated with the same transaction initiated by the client.
- GCOS 7 database updates by NatStar TPRs or by other GCOS 7 applications may be lost. See Section 4.3.2 *Avoiding Lost Updates* later in this chapter.

The result in either case will be inconsistent values across the database(s).

IMPORTANT:
If a NatStar client application initiates multiple RPC requests to one or more servers that are part of a single logical transaction, the application design must be structured to deal with a failure on a server ***after*** previous RPCs have resulted in committed database updates.

The following subsections describe some techniques for managing data integrity.

4.3.1 Combining Service Requests from the Client

TP applications often require the user to "navigate" through a series of screens and interactions with the host to accomplish tasks. You can hide those interactions in NatStar client applications, because you can use the GUI of the client application to accomplish the same results.

However, if a single interaction with the user results in multiple RPC interactions with GCOS 7, the following mismatch occurs:

- The user thinks that the single interaction represents a single transaction, **but**
- GCOS 7 processes multiple commitment units; i.e., multiple transactions.

In this situation, there is no automatic rollback of prior updates in the event of a client or server failure. The client application must then be able to "undo" those partial updates if any one of them fails. Careful design of the application as a whole is required to ensure that this "undo" operation is possible.

IMPORTANT:

When designing the client interaction with the user and with the server, you need to consider whether the user might become confused about how much processing has been completed successfully. If the user becomes confused, he or she may initiate a repeat of an interaction that has already caused update(s) to database(s). Careful design of the client application, and positive feedback to the user in all situations, can help to avoid user confusion.

A preferred approach for initiating several services is to combine multiple service requests into one client/server interaction, so that all the GCOS 7 database activity occurs within a single transaction. Multiple updates in a single transaction will allow rollback in case of failure of the TPR or GCOS 7, and will protect integrity by means of the standard GCOS 7 integrity-control mechanisms.

NOTE:

GCOS 7 integrity controls, including rollback, do not protect against inconsistencies that can occur in the case of a client failure after database updates are complete on GCOS 7. Refer to Section 4.3.3 *Updating Multiple Databases* later in this chapter.

4.3.2 Avoiding Lost Updates

Each RPC call/response constitutes a commitment unit on GCOS 7. Therefore, the client application cannot assume that data retrieved from a GCOS 7 database in one RPC call will remain unchanged until a subsequent RPC call. If a client application makes that assumption, the result can be a lost update, as illustrated in Figure 4-2.

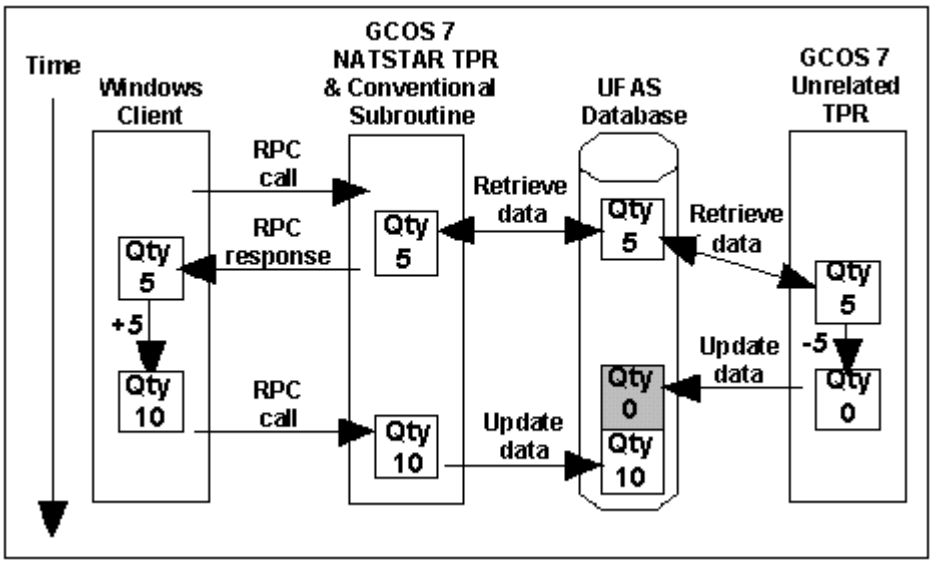

Figure 4-2. Lost Update

In the example in Figure 4-2, the first RPC call to the GCOS 7 server TPR results in a retrieval from the database. Because the data read by a NatStar TPR is released at the end of that procedure, another unrelated TPR is then able to read the same quantity and update it. Meanwhile, the client application is preparing an update against the same original quantity (5), then passing the result of that update back to the server TPR in a second RPC call. When the server TPR then updates the database without checking to see if the quantity is the same as when originally read, the update made by the unrelated TPR is lost. The result is an inaccurate value in the database.

IMPORTANT:
Figure 4-2 is a good illustration of how ***NOT*** to structure the logic of a client/server application.

The problems occurring in Figure 4-2 illustrate two important design concepts:

1. The importance of isolating business logic from user presentation. In this example, the client should simply forward a request to "add 5" to the business logic in the NatStar TPR. The TPR can then either fulfill the request or return a refusal, based on the value in the database.
2. The importance of avoiding the use of multiple client/server interactions to accomplish a single logical operation. Structuring the application so that each logical operation is packaged in a single client/server interaction is the optimum approach.

4.3.3 Updating Multiple Databases

One of the basic problems in handling updates against distributed databases is to ensure that related updates are synchronized: that is, either all of the updates are made or none are made. The mechanism typically used to ensure synchronization is the X/Open Distributed Transaction Processing (DTP) model. However, GCOS 7 does not implement DTP, and so care must be taken to avoid situations in which you need to synchronize multiple updates.

GCOS 7 Integrity Control *will* automatically synchronize updates to multiple GCOS 7 databases. GCOS 7 Integrity Control can also synchronize updates on GCOS 7 with remote updates initiated by GCOS 7. However, Integrity Control will *not* synchronize remote updates initiated by a Bull/NatStar client; for example, if a client initiates updates both on GCOS 7 and on a UNIX server and/or on the client.

As a result, you should avoid situations in which databases on both GCOS 7 and the client system or another server system would be updated. Any situation in which both a GCOS 7 server system and another server system (e.g., UNIX) are involved requires special care to ensure that database updates are handled correctly, as explained in the following discussion.

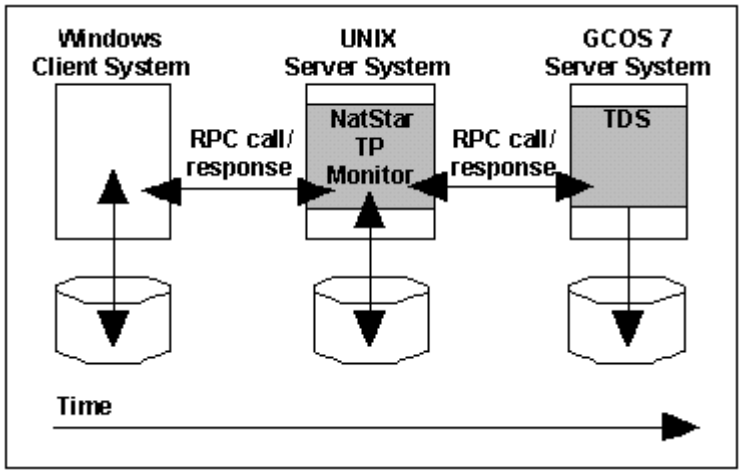

Figure 4-3. Updating Multiple Databases

Any GCOS 7 TPR that updates a database must be the last server invoked during the "prepare to commit" phase (in the DTP protocol) of a distributed database update to ensure consistency of the databases in the event of a failure by the client, any of the servers, or the NatStar TP Monitor. The GCOS 7 TPR must include logic to notify the calling application on UNIX of any failure in updating the database. The NatStar TP Monitor can then abort preceding transactions and roll back the associated updates in the "prepare to commit" phase. For more information, refer to the *NatStar Development Guide - Client-Server* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

The TUXEDO transaction monitor also conforms to the DTP model. TUXEDO therefore operates as described above.

IMPORTANT:

If a GCOS 7 system is involved in any client/server situation in which two or more databases (or files) are updated as the result of what is logically a single transaction, the GCOS 7 system must be the *last* system in the chain of updates. Also, the client and/or server code in the other system(s) involved must include logic to rollback related updates if the GCOS 7 system fails to complete its updates and return an acknowledgment of successful completion.

4.4 Implementing Security Controls

Methods Used in Conventional TDS Applications

To connect a conventional TDS, GCOS 7 typically requires USER, PASSWORD, PROJECT, and BILLING to check security.

Due to the NatStar/GCOS 7 architecture, it is necessary to respect the sequence:

- CONNECTION PHASE
- WORKING SESSION (EXCHANGES)
- DISCONNECTION PHASE

The pseudo server (NSCOM7 or OPEN 7 Gateway) acts for each user as a connection handler, submitting the connection message to the TDS mailbox, returning the status connected or not connected, and (if connected) memorizing the handle of the user. Then, when it receives a request message, it knows for which user it has to submit the request message. To terminate the working session, you must disconnect the user from the TDS. The connection and disconnection phase are mandatory in the application control flow.

An example of the implementation of connection and disconnection phases is done by using the templates provided with the NCL application code in:

- G7USRPASS for the NSCOM7 library,
- GCOS7_LINK library for NSCOM Light

To use these templates in your NCL application code, refer to Appendix D *GCOS 7 Configurations*.

Encryption Using NSCOM7

Bull/NatStar does not support any form of private-key or public-key encryption of messages between the client and server. However, the client/server applications can encrypt the parameters exchanged, if desired, by using encryption software that is not part of Bull/NatStar. (Encryption of the password is planned for the next release of the NatStar/GCOS 7 integration product.)

Encryption Using NSCOM Light

Going from the PC to the OPEN 7 Gateway, the password is encrypted automatically (this is the default option). However, if the user requests otherwise, in the configuration file inside the client on the PC, the password is not encrypted.

4.5 Step-by-Step Implementation Guide

This section describes briefly each of the steps required to implement a client/server application to operate with TDS on GCOS 7. This guide serves as an introduction to the more detailed implementation information in Chapter 5 *Implementing Client/Server Applications*.

For simplicity, this information is presented as a step-by-step process, as shown in Figure 4-4. In practice, some steps will occur in parallel, and some steps may be performed in a different sequence than that shown. The figure includes a reference to where each step is discussed in more detail. Most of the referenced discussions are either in Chapter 5 *Implementing Client/Server Applications*, otherwise they are in the appendices.

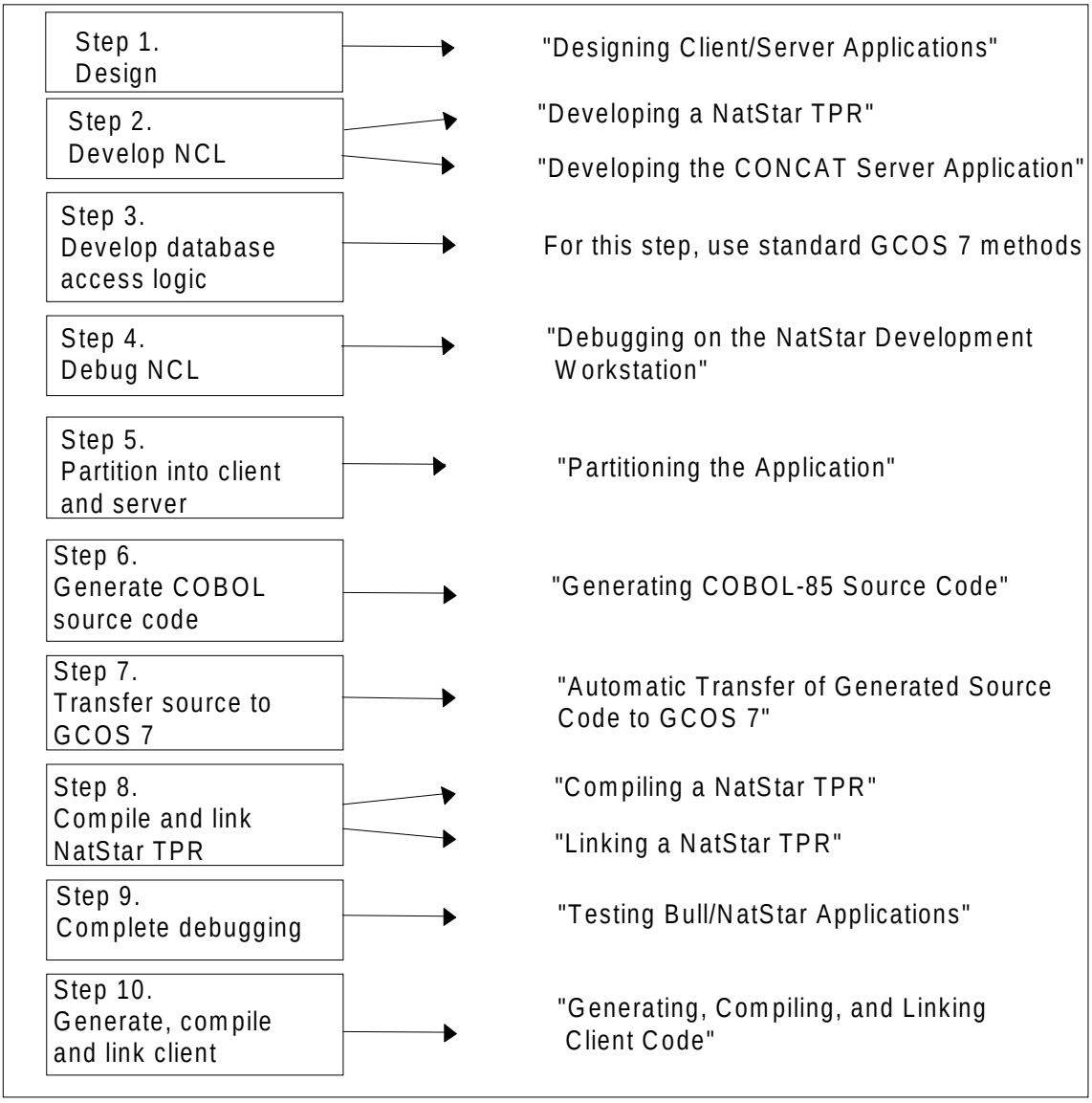

Figure 4-4. Step-by-Step Implementation Process with Cross-References

NOTE:
The appendices, which provide an example of how to develop a simple client/server application for PC and GCOS 7, illustrate graphically how to accomplish many of the steps shown in Figure 4-4.

IMPORTANT:
The following discussion assumes that the necessary hardware, software, and network facilities have been installed as described in earlier in this guide and in the *Bull/NatStar Software Release Bulletin.* This discussion also assumes that the required TDS(s) have been configured as described in Chapter 3 *Planning for the Bull/NatStar Environment*.

Step 1. Design the client/server application

Step 1 consists of the usual application-design process. Design decisions made early in the implementation cycle can have a significant impact on the success of the client/server application. Factors that are of special importance when designing a client/server application include:

- Deciding how to partition application logic across the client and server platforms.
- Deciding which part of the logic to implement using the NatStar development environment, and planning how to integrate the NatStar-generated logic with conventionally coded logic on GCOS 7.
- Evaluating the factors -- e.g., performance, security, data integrity -- discussed earlier in this chapter.

As usual, some parts of design may take place in parallel with the following steps. For example, prototypes of the user interfaces may be created as part of the definition of requirements.

Step 2. Develop the client and server logic in NCL on the NatStar development workstation

The next step consists of using a NatStar development workstation to create the NCL source for the client/server application. Step 2 consists of:

- Defining the user interface to the client/server application by designing one or more windows and the associated behaviors. The user interface defined in this way is platform-independent, and is only adapted later to the target hardware and software platform of the client systems.
- Specifying the logic to be included in the client application and in the server application. This logic is expressed in NCL, the NatStar Control Language. You may need to call conventional subroutines on GCOS 7 and/or use native (inline) code to express operations that cannot be expressed in NCL.

The output of Step 2 is a set of NCL source code, and possibly embedded native code, that includes all of the processing logic necessary for the client/server application *except* for logic that has been, or will be, implemented as conventional code on GCOS 7.

Step 3. Develop database-access logic on GCOS 7

In the current release, NatStar-generated code can access an Oracle database and UFAS files on GCOS 7, through a set of UFAS file access APIs described in detail in Section 5.3.1 *Using UFAS File Access APIs in NCL* in Chapter 5 *Implementing Client/Server Applications*.

Step 4. Debug the client/server NCL

At this point, the client and server processing logic has not been partitioned, but is defined in NCL that is organized in separate, but associated, libraries. You can debug this NCL interpretively on the NatStar development workstation.

However, the NCL developed in Step 2 does not include database-access logic, so you cannot debug the entire application. One way to extend the scope of Step 4 debugging is to simulate the GCOS 7 database-access logic in NCL. Other options for debugging are discussed later in this step-by-step guide and in Chapter 5 *Implementing Client/Server Applications*.

Step 5. Partition the NCL source into client and server portions

You next identify the NCL logic (library) that is to be exported to GCOS 7, which effectively partitions the application into client (PC-local) and server parts.

Step 6. Generate server source code

Steps 6 and 7 are done automatically by the BUILD function of NatStar.

Generate the NCL and the COBOL-85 code for the server part of the application -- that is, the part to be exported to GCOS 7. The NCL generator automatically creates the RPC stubs necessary to manage the client/server interaction, (the stubs structure is explained in Section 5.1.2 *NatStar TPR Structure and Processing Flow* in Chapter 5 *Implementing Client/Server Applications*).

Step 7. Automatic file transfer of the server COBOL source to GCOS 7

The file-transfer utility such as FTP (File Transfer Protocol) is used inside NatStar to automatically transfer the server source code to GCOS 7.

IMPORTANT:
Before you can compile and link the server source in Step 8 below, the NatStar specific software must have been installed on OPEN 7.

Step 8. Compile and link the NatStar TPR

Compile the server source code, using the standard COBOL-85 compiler.

NOTE:

The COBOL-85 source code produced by the NatStar code generator does not require any post-generation changes or hand-coding. COPY files that are required for compilation are provided by the NatStar code generator. COPY files in a GCOS 7 library must be of type CBX.

When the COBOL-85 source has been compiled successfully without fatal errors, link the compiled source to create a NatStar TPR.

If the source code includes calls to user libraries for access to UFAS files, or calls to other subroutines, link those libraries and/or subroutines with the compiled source.

IMPORTANT:

Before you can debug the complete client/server application in Step 9 below, the necessary client hardware and software must have been installed and the TDS(s) must have been configured, as noted in Chapter 3. Also, the TDS configuration must be capable of handling the new application.

Step 9. Complete debugging

The goal in Step 9 is to debug the entire client/server application using the target client platform (e.g., Windows) and the target server platform (GCOS 7).

You can debug the server source on GCOS 7 using PCF.

Step 10. Generate, compile, and link the client code

This step assumes that you have completed debugging and testing of the client/server application while executing the client code interpretively. Before you distribute copies of the client to client systems for production operation, you need to generate NCL and C source code, then compile and link the source to create an executable (`.EXE`) and Dynamic Link Library (`.DLL`).

Summary

Step 10 completes the process of designing and implementing the new client/server application. You can now prepare for production operation of the new application; e.g., by propagating the client software to all client systems.

Using the Appendices to illustrate this step-by-step implementation guide

At this point, you may want to work through the example given in the appendices. Appendix A explains how the example is presented. The appendices contain a simplified session during which you develop a client/server application. This session illustrates the operations that are performed on the NatStar development workstation. You can install the NatStar software on a PC and follow the description in the appendices to develop a simple application. This exercise can help you to understand the more detailed material provided in Chapter 5 *Implementing Client/Server Applications*.

5. Implementing Client/Server Applications

This section describes how to design and implement client/server applications in which client applications on Windows PCs access NatStar TPRs on GCOS 7. The focus in this section is on how to implement NSCOM7 server TPRs to operate under TDS on GCOS 7. Other aspects of client/server implementation are discussed as necessary to provide context.

IMPORTANT:

This chapter does not contain all of the information you need to develop a client/server application. Detailed information about the use of NatStar is provided in the Nat Systems documentation, and detailed information about the development of TDS TPRs is provided in other Bull documents (document references are listed in *Bibliography* in the *Preface* of this Guide). The purpose of this chapter is to provide:

a) a perspective on how the NatStar and GCOS 7 environments are used cooperatively;

b) detailed information about development requirements or techniques that are unique to the Bull/NatStar environment.

The following topics are included in this chapter:

- The NSCOM7 environment, which provides some general information about how NatStar and Bull software work together to support a client/server application.
- Special considerations when developing a NatStar TPR, which summarizes the major factors involved in developing TPR logic in NCL using a NatStar development workstation. This summary serves as an introduction to the detailed discussions of the same factors in the following section.
- Developing a NatStar TPR, which discusses each of the major factors involved in development.
- Generating server source code, which explains how to use the NatStar COBOL-85 code generator to create GCOS 7 source.

- Compiling a NatStar TPR, which describes how to compile the source generated by NatStar.
- Linking a NatStar TPR, which describes how to link the compiled code to create a TDS transaction.
- Testing Bull/NatStar applications, which discusses how to test the GCOS 7 code generated by NatStar and how to test the complete client/server application.
- Generating, compiling, and linking client code, which summarizes the steps necessary to prepare the client code for production operation.

5.1 The NSCOM Environment

To introduce the material in the rest of this chapter, this section provides:

- An overview of how NatStar software and Bull software are used together to create a client/server application, using NSCOM7 (NS02COMT driver) or NSCOM Light (NS02OP7G driver).
- An explanation of the structure and processing flow of a NatStar TPR.

5.1.1 The Bull/NatStar Development Environment

The environment in which you will develop Bull/NatStar client/server applications consists of the following components:

- NatStar Development Workstation.
- TDS environment.
- NSCOM7 or NSCOM Light communication link.

The remainder of this section describes how to use each of these components.

5.1.1.1 Using the NatStar Development Workstation

You can develop a complete client/server application on a NatStar development workstation. You define both the user interface (windows, buttons, etc.) and the associated client *and* server logic, which is written in NCL (the Nat Systems Control Language). You can cause client NCL (NCL that will be executed by the client in the completed application) to use an RPC (Remote Procedure Call) to invoke server NCL simply by coding an NCL statement that references the name of a server function or instruction. Refer to the discussion in Section 2.1.2 *Using Remote Procedure Call* in Chapter 2 *Bull/NatStar in the Client/Server Environment*.

In NatStar, instructions and functions are defined in NCL and organized in libraries. In NCL, a function returns a value that is usually a return status, while an instruction does not return a value. This distinction is important in NCL, but does not exist in COBOL.

NOTE:

To avoid continual references to "instructions or functions" in the following general discussion, the term "instruction" is used to imply either an instruction or a function. Where the distinction between instruction and function is important, the difference is specified explicitly.

Basic Guidelines for Using Libraries

This section explains how to use NatStar libraries when developing a Bull/NatStar client/server application.

You must create at least two NatStar libraries for the client/server application; one (or more) for a client, and one (or more) for the server.

IMPORTANT:

Client logic and server logic must be packaged in separate libraries. The server logic cannot be exported to GCOS 7 if it is contained in the same library as client logic.

- The client library or libraries are used to generate code that will execute on the client; for example, code generated in C Language for Windows. Client libraries can contain both graphical resources (windows, buttons, text boxes, etc.) and NCL resources (instructions, segments, constants, etc.).
- The server library or libraries are used to generate code that will run on the server; for example, COBOL-85 code that will become a TDS transaction. Server libraries contain only NCL resources; they cannot contain any graphical resources. A library is generated corresponding to each TDS transaction.
- Instructions in libraries are declared (by using a checkbox) as either Local or Remote.

  Client instructions are always declared to be Local.

  Server instructions can be declared to be either Local or Remote. The significance of declaring an instruction Remote is that it will be executed on GCOS 7 when invoked by a client. For a more detailed explanation of when to declare a server instruction as Remote, refer to Section 5.4.1 *Background Information* in Section 5.4 *Generating Server Source Code* later in this chapter.

- Client instructions can call other client instructions. Client instructions can also call server instructions.
- Server instructions can call other server instructions. However, Bull/NatStar server instructions cannot call NCL instructions on other systems or other libraries.
- All of the server instructions contained in a single server library will be compiled to form one TDS transaction -- they cannot be separated into individual TPRs. When a client calls a server instruction on GCOS 7, the NSCOM software automatically executes the correct instruction within the transaction.
- All functions in a server library must be grouped in a single transaction. It is not possible to call an NCL function from another library.
- A single client can call remote server instructions that exist in multiple transactions, but each such call requires a separate RPC. The called TPRs may or may not be in the same TDS, and may not even be running on the same GCOS 7 system. Each call is a separate transaction, and each call results in committed database updates if any updates are performed.

The example client/server application in the appendices illustrates how to create libraries and how to associate client and server libraries.

5.1.1.2 Using the TDS Environment

TDS is unaware that it is running Bull/NatStar server transaction. A Bull/NatStar TDS uses the same TDS software as all other TDSs. A Bull/NatStar TDS receives an input message and processes it, just as other TDSs do. Messages sent by the transaction are returned to the source of the input message.

In Bull/NatStar, however, the input message is created automatically on the client when the client NCL calls a server instruction. Because the application developer does not explicitly create this input message, the developer need not be aware of the specific content or format of the message, but only of the application-specific content of the message.

Transaction code generated automatically by NatStar receives the input message, extracts the server instruction name and arguments, and generates a call to the server instruction.

When the server instruction completes execution, transaction code automatically generated by NatStar places the implicit return value (if applicable) and the return arguments into an output message, then sends the message to the calling client program. Because the application developer does not explicitly create this output message, the developer need not be aware of the specific content or format of the output message.

5.1.1.3 Using the NSCOM Communication Link

When a client invokes a server instruction by naming it in a client NCL statement, the client knows the name of the instruction. However, the instruction name alone does not provide the location of the server TPR -- i.e., the GCOS 7 IP address. Therefore, the developer of the client NCL must also specify the location, name of the server, and the transaction name in the:

- NSCOM.INI file for NSCOM7.
- Virtual Node for NSCOM Light.

Also, the server name can be changed dynamically, while the client application is operating, if the developer codes this into the client NCL. For example, if a client calls instructions from two independent server libraries (that were used to create two TPRs), the client application would need to switch between two server names as calls were made. As noted earlier, the two TPRs could be installed in two TDSs that might be running on two different GCOS 7 systems. Also, some server instructions could reside on non-GCOS systems, such as UNIX servers.

Specifying the Server Name

NatStar provides special NCL statements that are used to specify the server name. These statements are described fully in the *NatStar Development Guide -- Client-Server* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

For example, one of the easiest ways to specify the server name in NCL is to code the `THINGS_SET_DEFAULT_SERVER` statement in the `INIT` event for the main window of the client application. After the server name has been specified in this way, all server instruction references (RPCs) will be sent to the server identified by this statement. The format of the statement is:

```
THINGS_SET_DEFAULT_SERVER "server_name"
```

Refer to the example application in the appendices for an example of how to use this statement.

If a server name has not been defined (e.g., `THINGS_SET_DEFAULT_SERVER` has *not* been executed), NatStar attempts to find and execute the remote server instruction locally on the client system, even though the instruction has been declared to be remote. This feature can be useful during testing, because client and server code can be tested on the NatStar development workstation (provided that access to GCOS 7 databases is not required or can be simulated).

Locating the Named Server

NSCOM invokes an instruction on a remote server when the following conditions are met:

- The server name has been defined (as described above).
- The client NCL references an instruction defined in a server library as remote (i.e., its Remote box is checked).
- The communication path from the client to the server has been configured correctly and is operational. On a client PC, the communication software is configured using a set of configuration files. The following discussion is based on the use of those configuration files for NSCOM7. For NSCOM Light, see Chapter 8 *How to Use the NSCOM Light (NS02OP7G).*

To locate the named server, NSCOM7 uses the following procedure.

1. NSCOM first searches the `[CLISERVICES]` section of the `NSCOM.INI` file for a `SERVICE=` entry whose value matches either the specified *`server_name`* or a special service (described below) named `SRVNTFND`. The search can have one of the following results:
   a. `SERVICE=`*`server_name`* is found. NSCOM processing continues as described in step 2 below.
   b. `SERVICE=`*`server_name`* is *not* found, but a `SERVICE=SRVNTFND` entry is found. NSCOM processing continues as described in Step 2.

      If `SERVICE=SRVNTFND` is the **only** `SERVICE=` entry, then **all** remote instructions will use this entry. This fact is useful when the client uses only one TDS server.

   c. Neither `SERVICE=`*`server_name`* nor `SERVICE=SRVNTFND` is found. The server instruction call fails and an error is generated.
2. If a "match" is found in Step 1a or Step 1b, NSCOM then examines the `[CLISERVICES]` entries that follow the located `SERVICE=`*`server_name`* or `SERVICE=SRVNTFND` entry, searching for:
   a. `DESTSERVICE=`*`TDS_name`*. If this entry is present, the specified *TDS_name* is used as the TDS name.
   b. `REMOTENODE=` *IP_address* of the server on the Network.
   c. `CPLDESTSERVICE=` *transaction_name* representing the name of the transaction to be submitted by the dedicated NSERV7 pseudo server, launched for the specified TDS on the GCOS 7 remote node.

      If any of this information is not set in the NSCOM.INI file, the server instruction call fails and an error is generated.

5.1.2 NatStar TPR Structure and Processing Flow

As explained in the preceding discussion, server libraries are used to generate NatStar TPRs. This section describes the structure of a NatStar COBOL-85 TPR, and explains how the various software components work together. This is also illustrated by Figure 5-1. This figure also illustrates some of the fundamental differences between NatStar TPRs and conventional TPRs.

NOTE:

The information in this section is useful if you want to study the COBOL code generated by NatStar. You do *not* need this information simply to develop a Bull/NatStar client/server application.

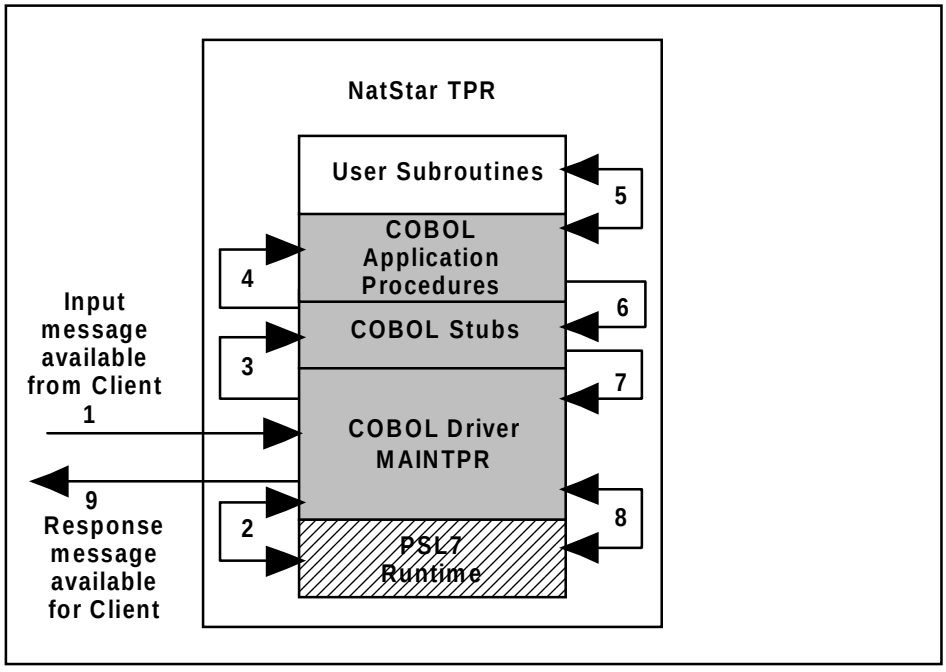

Figure 5-1. Structure and Processing Flow of a NatStar TPR

NOTE:

The shaded areas (COBOL...) in Figure 5-1 represent code that is generated by the NatStar development workstation. The patterned area (PSL7 Runtime) represents the runtime routines that are bound with the NatStar TPR. The unshaded area represents conventional (non-NatStar) code that may or may not be used.

Figure 5-1 illustrates how a NatStar server transaction is invoked, and shows the flow of processing among the software modules.

The numbered arrows in Figure 5-1 have the meanings explained below.

1. The input message from the client contains the fields shown in Figure 5-2. The input message contains the name of the called server instruction or function, and its parameters, as described earlier in Section 5-4 *Using the TDS Environment*.

Transaction Name	Instruction or Function Name	Parameter-1	Parameter-2	. . .	Parameter-n

Figure 5-2. Content of Input Message to TPR

   TDS accepts the input message and invokes the NatStar TPR configured for that transaction. Control is given to the TPR through the COBOL Driver (transaction Driver), MAINTPR. MAINTPR is generated automatically for the NatStar TPR when COBOL code is generated by the NatStar code generator.

2. MAINTPR calls the PSL7 NSCOM runtime routines to receive the input message. PSL7 NSCOM executes the RECEIVE necessary to obtain the message from TDS.

3. When control is returned by PSL7 NSCOM, MAINTPR evaluates the instruction or function name specified in the input message (see Figure 5-2) and invokes the COBOL stub that corresponds to that instruction or function. The COBOL stub is generated automatically by the NatStar code generator.

   If the NCL library that defines the NatStar TPR includes a single instruction or function, the NatStar TPR consists of a single stub and procedure. If the NCL library includes multiple instructions or functions, the NatStar TPR includes a stub plus procedure for each instruction or function.

4. The invoked COBOL stub extracts the parameters from the input message and calls the corresponding COBOL application procedure, passing it the parameters from the input message. The COBOL application procedure consists of the TPR logic that was developed in NCL on the NatStar development workstation.

5. The application procedure can optionally call conventional user subroutines, which were not generated using NatStar (only if the subroutine doesn't call any SEND on the user input terminal).

6. When the application procedure completes execution, it returns control to the COBOL stub, passing the output parameters that were created by the application procedure. These parameters will form the output message to be sent to the client.
7. The COBOL stub places the output parameters in the message buffer, then exits to MAINTPR.
8. MAINTPR calls the PSL7 NSCOM runtime routines to send the response to the client. PSL7 NSCOM executes the SEND necessary to release the output message to TDS.
9. On return from PSL7 NSCOM, MAINTPR returns control to TDS, which completes the execution of the NatStar TPR.

5.2 Special Considerations when Developing a NatStar TPR

This section summarizes the topics that require special consideration when you are designing and implementing a NatStar TPR. All of these considerations relate to the process of developing transaction logic in NCL on a NatStar development workstation.

NOTE:
You also need to consider each of the general guidelines for client/server application design that are discussed in Chapter 4 *Designing Client/Server Applications*.

Each of the topics listed below is discussed in more detail in the following paragraphs commencing with Section 5-13 *Developing a NatStar TPR*. There is a paragraph that has the same title as the name of each topic listed in this section. For example, the topic *Handling Data-Type Incompatibilities* below is described in more detail later in this chapter in Section 5-47 *Handling Data-Type Incompatibilities*.

IMPORTANT:
NatStar generates a transaction with one TPR only and one Commitment Unit only.

Using Packed-Decimal Data Type in NCL

A limited version of support for packed-decimal data is included in the current NatStar release.

Handling Data-Type Incompatibilities

NatStar TPRs are coded in NCL. As a result, a NatStar TPR can use only NCL-compatible data types, which represent a subset of the data types supported in COBOL on GCOS 7.

Whenever a NatStar TPR calls a COBOL subroutine on GCOS 7, you need to determine if:

1. The called subroutine expects to receive a data value expressed in a data type that is not NCL compatible.
2. The called subroutine returns a data value expressed in a data type that is not NCL compatible.

You need to include logic in the NatStar TPR or in conventional code to handle these incompatibilities.

Using Native Code

When developing a NatStar TPR using NCL, you may find that you cannot express a needed operation in NCL. In this situation, you can use the NCL native-code feature (also called "inline code") to include COBOL-85 code in your NCL program. Note that the use of native code limits the portability of code developed in NCL. Often you can use other techniques, such as calling a conventional subroutine on GCOS 7, to provide the required operation.

Calling a Conventional Subroutine

A NatStar TPR can access GCOS 7 UFAS files and an Oracle database. Also a NatStar TPR can indirectly access a GCOS 7 database by calling a conventional subroutine.

IMPORTANT:
When you call a conventional subroutine from a NatStar TPR, you need to be concerned about data-type incompatibilities.

Handling Errors

Various types of errors that are related to the Bull/NatStar client/server environment can occur during the processing of a NatStar TPR. You need to include logic to handle these errors in the client application.

Avoiding COBOL Reserved Words in NCL

The NatStar COBOL generator contains a limitation that you can avoid as follows. Avoid the use of COBOL reserved words as:

- the names of NCL instructions.
- the names of NCL functions.
- the names of NCL parameters; i.e., variables to be listed in the LOCAL clause.

Avoiding Recursion in NCL

Do not use recursive calls in NCL that will be code generated in COBOL and become server logic on GCOS 7.

COBOL does not support recursion. However, neither the NatStar code generator nor the COBOL-85 compiler on GCOS 7 will identify and flag recursion in COBOL source. The result when the object program is executed can be a loop that repeats for an infinite number of times, or possibly a program abort. You can avoid these problems by being sure not to use recursion in NCL that will be generated in COBOL.

NOTE:

The last two items above (*Avoiding COBOL Reserved Words in NCL* and *Avoiding Recursion in NCL*) are fully covered above and not therefore discussed further in this chapter.

5.3 Developing a NatStar TPR

The following paragraphs describe how to implement a NatStar TPR that will operate as a server (service) in the Bull/NatStar environment. They provide a detailed discussion of the topics introduced earlier in this chapter.

5.3.1 Using UFAS File Access APIs in NCL

These APIs are described in the file NS02UFAS.NCL delivered with the NatStar/GCOS 7 integration product, in the OPEN 7 partition `/usr/natsys/appli/ns02ufas.ncl`.

This paragraph describes the available UFAS entry points. Each entry point is known in NCL as an instruction.

IMPORTANT:
All possible return codes from UFAS are listed, but for information purposes only. Certain codes should never appear since they do not correspond to a possible situation with Bull/NatStar. For full details of UFAS aspects, see the *UFAS Extended User's Guide*.

5.3.1.1 NSUFAS_LAST_ERROR

This instruction returns the last error code for the last UFAS instruction executed in this source code. Use this instruction to test for completion of the UFAS primitive.

Parameters

INT RET_STATUS (4) The error code returned for the last UFAS primitive.

For more information about the value and the meaning of the return code, see the manual *Messages And Return Codes Directory*.

A normal value is in the range of 0 to 255:

- The value 0 indicates successful completion of the UFAS primitive. The return code DONE has the value 0.
- Values between 1 and 255 indicate normal completion of the UFAS primitive, and indicate a functional state of the primitive.
- For example, the return code SPLIT has the value 233 (hexa E9) which means that an internal split has been performed by the UFAS access method.

- Values beyond 255 indicate abnormal execution of the UFAS primitive. For example, the return code RECNFD has the decimal value 1079 (hexa 0437) which means that the record is not found or has been deleted.

Return Codes

Return codes from UFAS are in the form XUFAS (RC=xxxxxxxx->XUFAS xx,value).

To test a Return Code, you must refer to the following rule:

1. Install, in NatSar, the NS02G7RC service for the COBOL Library and for the libraries.

   The NS02G7RC.NCL file contains the list of GCOS 7 Return Code value. These Return Code can be referenced directly from NCL code for NatSar 2.15.

2. When you test the Return Code, be sure that it is not belong to a class. You can recognize a Return Code class by the suffix which is _A for the beginning value of the class, and _Z for the ending value of the class.

3. You can write in NCL:

   a. When you test a Return Code:

   ```
   Local int Return_code (2)
   If Return_code = G7RC_value% ...
   ```

   b. When you test a Class of Return Code:

   ```
   Local int Return_code (2)
   If (Return_code >= G7RC_value_A%) and
                      (Return_code <= G7RC_value_Z%) ...
   ```

EXAMPLE:

To test the return code SPLIT, you can write:

```
local int Return_code (2)
if Return_code = G7RC_SPLIT%
   ...
else
   ...
endif
```

❑

When they originate from GPLP, the module that prepares the call to UFAS, (RC=xxxxxxxx->GPLP 0, value), the following values can be returned :

CALLVIOL	The primitive was called with a wrong number of arguments.
IFNERR	The file handle does not lead to valid file structures.
ARGERR	An argument (PSIZE) cannot be reached.
WRONGPAR	The file handle cannot be reached for NSUFAS_GET_FILE_HANDLE.
ADDRERR	The PBUFFER parameter cannot be reached.
KEYERR	The PINKEY or PINADDR parameter cannot be reached.
OPTERR	An argument (PCOND, PKEYID) cannot be reached.
INVUSE	Bad parameter addressing.

5.3.1.2 NSUFAS_LAST_ERRMSG

Returns the last error message (the error message associated with the last error code) of the last UFAS primitive used.

Parameters

CHAR ERRMSG (70)	The error message returned for the last UFAS primitive used.

5.3.1.3 NSUFAS_GET_FILE_HANDLE

Gets the file handle for the named file (this instruction returns the addressability of a file). This instruction must be used prior to the access method instructions.

Parameters

POINTER PFHANDLE	The file handle returned associated with the IFN.
CHAR FNAME$(8)	The input file name of the IFN assigned in the STDS.

Return Codes

DONE	The PFHANDLE contains the file handle associated with the IFN.
NOMATCH	The IFN is unknown.

5.3.1.4 NSUFAS_READ

Reads the next record of a file (sequential, indexed sequential, or relative).

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	Amount of input data to be read.

Normal Return Codes

DONE	NSUFAS_READ has been successfully executed.
COUNTLIM	NSUFAS_READ has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
DATALIM	NSUFAS_READ has failed because there is no next record to retrieve in the file.
VOLLIM	The NOFORCE option has been declared in the FD and the end of the current volume is reached. The execution of the NSUFAS_READ primitive is unsuccessful.
WALIM	NSUFAS_READ has been successfully executed, but the returned record is truncated to PSIZE bytes.

NOTE:
If two normal conditions occur at the same time, only one is reported by a normal return code, with the following priority: DATALIM, VOLLIM, WALIM, COUNTLIM.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several Sequence Control Blocks (SCBs), each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in SEQUENTIAL access mode but an NSUFAS_READ_RELATIVE instruction was issued.
EXHAUST	A DATALIM return code has already been returned on a previous instruction referencing the same SCB.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
LNERR	The specified length is negative or null.
NOCURREC	NSUFAS_READ was issued but the current record pointer is not valid. The previous NSUFAS_READ or NSUFAS_START primitive using the same SCB has been unsuccessful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.

RECNFD	The record address given does not correspond to an existing record in the file.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened with OUTPUT or APPEND processing mode.

5.3.1.5 NSUFAS_READ_INDEXED

Reads an indexed file with a specified primary key. If you want to specify a subkey, then the PKEY must point to a primary key with the low-value byte (hexa 00) in the unspecified part of the primary key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	Amount of input data to be read.
POINTER PKEY	The indexed key of the record. Note that the size of INKEY depends on the file.

Normal Return Codes

DONE	NSUFAS_READ_INDEXED has been successfully executed.
COUNTLIM	NSUFAS_READ_INDEXED has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
DATALIM	NSUFAS_READ_INDEXED has failed because there is no next (or no prior) record to retrieve in the file.
SYNONYM	NSUFAS_READ_INDEXED has been successfully executed and the next record according to the reference key is a duplicate of the current record. (It has the same key value as the current record.)

WALIM	The read has been successfully executed, but the returned record is truncated to PSIZE bytes.

NOTE:
If two normal conditions occur at the same time, only one is reported by a normal return code, with the following priority: DATALIM, WALIM, SYNONYM, COUNTLIM.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that this lack of buffer space can occur when using several SCBs, each of which can use up to two buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in sequential access mode when a NSUFAS_READ_INDEXED is issued.
EXHAUST	A DATALIM return code has already been returned on a previous instruction referencing the same SCB.
FLNAV	Attempt to access the file in DIRECT mode and the index tree is damaged.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
KEYERR	The PKEY parameter is not valid, no corresponding key is found.
LNERR	The specified length is negative or null.
NOTOPEN	The file is not open.

NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	The record key defined by PKEY does not correspond to an existing record in the file.
SCIDXNAV	Secondary index(es) is (are) damaged or not created.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened with OUTPUT or APPEND processing mode.

5.3.1.6 NSUFAS_READ_RELATIVE

Reads a relative file with the specified key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	Amount of input data to be read.
POINTER PINADDR	Address of the record in the relative file. (Note that the size of INADDR is INT(4) which contains the number of the record to be read. The first record is number 1.)

Normal Return Codes

DONE	NSUFAS_READ_RELATIVE has been successfully executed.
COUNTLIM	NSUFAS_READ_RELATIVE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

DATALIM	NSUFAS_READ_RELATIVE has failed because there is no next record to retrieve in the file.
WALIM	NSUFAS_READ_RELATIVE has been successfully executed, but the returned record is truncated to PSIZE bytes.

NOTE:

If two normal conditions occur at the same time, only one is reported as a normal return code, with the following priority: DATALIM, WALIM, COUNTLIM.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that this lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in sequential access mode when a NSUFAS_READ_RELATIVE is issued.
EXHAUST	A DATALIM return code has already been returned on a previous instruction referencing the same SCB.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
LNERR	The specified length is negative or null.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.

RECNFD	The record key defined by PINADDR does not correspond to an existing record in the file.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONCPMD	The file is opened in OUTPUT or APPEND processing mode.

5.3.1.7 NSUFAS_READ_COND

Reads a record in a sequential indexed file with a conditional read direction. Used for sequential indexed files only.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	Amount of input data to be read.
CHAR COND$ (2)	The read condition to find the record. COND$ can take the following values: "FI", read the FIrst record "NX", read the NeXt record "PR", read the PRevious record "LA", read the LAst record

Return Codes

Same as for NSUFAS_READ_INDEXED above.

5.3.1.8 NSUFAS_READ_SECKEY

Reads an indexed file with a specified secondary key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	Amount of input data to be read.
POINTER PKEY	The input pointer to the record key.
INT KEYID (2)	The input offset of the key to search for in the record, in bytes. The first byte is at offset 1.

Return Codes

Same as for NSUFAS_READ_INDEXED above.

5.3.1.9 NSUFAS_UPDATE

Updates a record in a sequential file.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer of data to be updated.
INT PSIZE (2)	The amount of input data to be updated (length (ALN)).

Normal Return Codes

DONE	NSUFAS_UPDATE has been successfully executed.
COUNTLIM	NSUFAS_UPDATE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
LNERR	The specified record length or SIZE parameter is different from the actual record size.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.10 NSUFAS_UPDATE_INDEXED

Updates the record in an indexed file with the specified primary key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer to be filled with data.
INT PSIZE (2)	The amount of input data to be updated (length (ALN)).
POINTER PKEY	The indexed key of the record. Note that the size of PKEY depends on the file.

Normal Return Codes

DONE	NSUFAS_UPDATE_INDEXED has been successfully executed.
COUNTLIM	NSUFAS_UPDATE_INDEXED has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
SPLIT	NSUFAS_UPDATE_INDEXED has been successfully executed and index reorganization has been done for one or more secondary keys.
SYNONYM	NSUFAS_UPDATE_INDEXED has been successfully executed and at least one secondary key has been modified for which duplicates are allowed and that has the same value as another existing record.

NOTE:

If two normal conditions occur at the same time, only one is reported as a normal return code, with the following priority: SYNONYM, COUNTLIM.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that this lack of buffer space can occur when using several SCBs, each one being able to use up to 2 buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in sequential access mode and the PKEY parameter is provided.
DUPKEY	A secondary key value is changed, for a key that does not allows duplicates, and a record already exists in the file with the same key value.
FLNAV	File contents are damaged.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
KEYCHG	An attempt is made to change the primary key value.
LNERR	The specified record length is different from the actual record size.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	The record key defined by PKEY does not correspond to an existing record in the file.
SCIDXNAV	Secondary index(es) is (are) damaged or not created.

UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.11 NSUFAS_UPDATE_RELATIVE

Updates a record in a relative file with the specified key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer of new data to update.
INT PSIZE (2)	The amount of input data to be updated (length (ALN)).
POINTER PINADDR	The input address of the record to update. Note that the size of the INADDR is INT(4) which contains the record number to be read. The first record is number 1.

Normal Return Codes

DONE	NSUFAS_UPDATE_RELATIVE has been successfully executed.
COUNTLIM	NSUFAS_UPDATE_RELATIVE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

ADDROUT	The record address is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
LNERR	The specified record length is different from the actual record size.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	The record key defined by INKEY and INKEYID does not correspond to an existing record in the file.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.12 NSUFAS_WRITE

Writes in a sequential file.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer of data to be written.
INT PSIZE (2)	The input number of bytes to be written (length (ALN)).

Normal Return Codes

DONE	NSUFAS_WRITE has been successfully executed.
COUNTLIM	NSUFAS_WRITE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
DATALIM	This return code is not applicable here.
VOLLIM	The NOFORCE option has been declared in the FD and the end of the current volume is reached. The execution of the NSUFAS_WRITE primitive is unsuccessful and a FORCE primitive must be issued to go on processing the next volume of the file.

NOTE:

If two normal conditions occur at the same time, only one is reported by a normal return code, with the following priority: DATALIM, VOLLIM, COUNTLIM.

Abnormal Return Codes

BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in DIRECT access mode and an NSUFAS_WRITE is used. (No INADDR parameter provided.)
EXHAUST	There is no more space in the file to put the new record.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
IOFAIL	A previous NSUFAS_WRITE has been unsuccessful due to an unrecoverable I/O error. File filling is no longer possible.
LNERR	The record length is invalid.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT or UPDATE processing mode.

5.3.1.13 NSUFAS_WRITE_INDEXED

Writes to an indexed file with the specified primary key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer of data to be written.
INT PSIZE (2)	The input number of bytes to be written (length (ALN)).
POINTER PKEY	The indexed key of the record. Note that the size of PKEY depends on the file.

Normal Return Codes

DONE	NSUFAS_WRITE_INDEXED has been successfully executed.
COUNTLIM	NSUFAS_WRITE_INDEXED has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
DATALIM	This return code is not applicable here.
SPLIT	NSUFAS_WRITE_INDEXED has been successfully executed and a local file reorganization may have been done. Any record in the file may have changed its Simple File Relative Address (SFRA).
SYNONYM	NSUFAS_WRITE_INDEXED has been successfully executed and at least one secondary key of the inserted record, for which duplicates are allowed, has the same value as an existing record. Note that, since secondary key are not processed when the file is opened in OUTPUT processing mode, SYNONYM can only be returned when the file is opened in UPDATE or APPEND processing modes.

NOTE:
If two normal conditions occur at the same time, only one is reported by a normal return code, with the following priority: DATALIM, SYNONYM, SPLIT, COUNTLIM.

Abnormal Return Codes

BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
DUPKEY	A record already exists in the file with the same key value for a key that does not allows duplicates. If the file is opened in OUTPUT processing mode, DUPKEY can only concern the primary key. If it is opened in UPDATE or APPEND processing mode, it can concern any key.
EXHAUST	The file is opened in OUTPUT processing mode and there is no more room in the file to insert a new record.
FLNAV	File contents are found damaged or left damaged by the PUT primitive.
FILEOV	The file is opened in UPDATE or APPEND processing mode and there is no more room in the file to make a local reorganization to insert the new record. Note that there can be some available space in the file allowing insertion of another record with other key values.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
IOFAIL	A previous NSUFAS_WRITE_INDEXED has been unsuccessful due to an unrecoverable I/O error. File filling is no longer possible.
LNERR	The record length is invalid.

NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
OUTSEQ	The file is opened in OUTPUT or APPEND processing mode and the primary key value of the inserted record is not greater than that of an existing record in the file.
SCIDXNAV	Secondary index(es) is (are) damaged or not created.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT processing mode, or with UPDATE mode and SEQUENTIAL access.

5.3.1.14 NSUFAS_WRITE_RELATIVE

Writes to a relative file with the specified key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PBUFFER	The input buffer of data to be written.
INT PSIZE (2)	The input number of bytes to be written (length (ALN)).
POINTER PINADDR	Address of the record in the relative file. (Note that the size of INADDR is INT(4) which contains the number of the record to be read. The first record is number 1.)

Normal Return Codes

DONE	NSUFAS_WRITE_RELATIVE has been successfully executed.
COUNTLIM	NSUFAS_WRITE_RELATIVE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.
DATALIM	This return code is not applicable here.

NOTE:
If two normal conditions occur at the same time, only one is reported by a normal return code, with the following priority: DATALIM, COUNTLIM.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
CONFLICT	The file is opened in DIRECT access mode and an NSUFAS_WRITE is used. (No INADDR parameter provided.)
DUPKEY	A record already exists in the location specified by the INADDR parameter in NSUFAS_WRITE_RELATIVE.
EXHAUST	There is no more room in the file to put the new record.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.

IOFAIL	A previous NSUFAS_WRITE_RELATIVE has been unsuccessful due to an unrecoverable I/O error. File filling is no longer possible.
LNERR	The record length is invalid.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT processing mode, or with UPDATE mode and SEQUENTIAL access.

5.3.1.15 NSUFAS_START_INDEXED

Sets the current record to the one specified with the primary key in an indexed file. If you want to specify a subkey, PKEY must point to a primary key with the low-value byte (hexa 00) in the unspecified part of the primary key.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PKEY	The indexed key of the record. Note that the size of PKEY depends on the file.
CHAR COND$ (2)	Conditions the start point for the record search. COND$ can take the following values. "EQ" the first record whose key is EQual to the input key. "GT" the first record whose key is Greater Than the input key. "GE" the first record whose key is Greater than or Equal to the input key

Normal Return Codes

DONE	NSUFAS_START_INDEXED has been successfully executed.
COUNTLIM	NSUFAS_START_INDEXED has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Normal Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
FLNAV	Index tree is damaged.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
KEYERR	The parameter PKEY is not valid, no corresponding key is found.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	There is no record in the file satisfying the specified condition.
SCIDXNAV	Secondary index(es) is (are) damaged or not created.

UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in OUTPUT or APPEND processing mode.

5.3.1.16 NSUFAS_START_RELATIVE

Sets the current record in a relative file.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PINADDR	The input address for setting the current working flow. (Note that INADDR is INT(4) which contains the number of the record to be read. The first record is number 1.)
CHAR COND$ (2)	Conditions the start point for the record search. COND$ can take the following values. – the first record whose key is EQual to the input key. – the first record whose key is Greater Than the input key. – the first record whose key is Greater than or Equal to the input key

Normal Return Codes

DONE	NSUFAS_START_RELATIVE has been successfully executed.
COUNTLIM	NSUFAS_START_RELATIVE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	There is no record in the file satisfying the specified condition.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.17 NSUFAS_START_BEGIN

Sets the current record to the first one of the file (sequential, sequential indexed, or relative).

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.

Return Codes

Same as for NSUFAS_START_INDEXED above.

5.3.1.18 NSUFAS_START_SECKEY

Sets the current record to the one specified with the secondary key in an indexed file. If you want to specify a subkey, then the PKEY must point to a secondary key with the low-value byte (hexa 00) in the unspecified part of the secondary key parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PKEY	Pointer of the input key.
INT KEYID (2)	Identifier of the record key.
CHAR COND$ (2)	Conditions the start point for the record search. COND$ can take the following values. – the first record whose key is EQual to the input key. – the first record whose key is Greater Than the input key. – the first record whose key is Greater than or Equal to the input key

Return Codes

Same as for UFAS_START_INDEXED above.

5.3.1.19 NSUFAS_DELETE

Logically deletes the current record in a sequential disk file.

Parameter

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.

Normal Return Codes

DONE	NSUFAS_DELETE has been successfully executed.
COUNTLIM	NSUFAS_DELETE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.

WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.20 NSUFAS_DELETE_INDEXED

Deletes the record specified with the primary key of an indexed file.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PKEY	The indexed key of the record. Note that the size of PKEY depends on the file.

Normal Return Codes

DONE	NSUFAS_DELETE_INDEXED has been successfully executed.
COUNTLIM	NSUFAS_DELETE-INDEXED has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
FLNAV	File contents are damaged.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.

INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	The record the key defined by PKEY does not correspond to an existing record in the file.
SCIDXNAV	Secondary index(es) is (are) damaged or not created.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.1.21 NSUFAS_DELETE_RELATIVE

Deletes the record specified by the key in a file.

Parameters

POINTER PFHANDLE	The input file handle returned by the instruction NSUFAS_GET_FILE_HANDLE.
POINTER PINADDR	The input address of the record to delete. Note that the size of INADDR is INT(4) which contains the record number to be read. The first record is the number 1.

Normal Return Codes

DONE	NSUFAS_DELETE_RELATIVE has been successfully executed.

COUNTLIM	NSUFAS_DELETE_RELATIVE has been successfully executed and the number of file accesses previously done corresponds to the number given in the CKPTLIM parameter of the associated FD. The counter is reset to 0.

Abnormal Return Codes

ADDROUT	The record address given is outside file limits.
BUFNBOV	The NUMBUF parameter value given in the FD does not correspond to the user's buffer usage. Request more buffers. Note that such a lack of buffer space can occur when using several SCBs, each one being able to use up to two buffers, or when sharing buffers between several files in a pool.
HEADERR	CI inconsistency detected after a read (integrity Header and Trailer do not match).
IFNERR	The file handle does not lead to valid file structures.
INDUNKN	The specified SCB identifier does not correspond to an allocated SCB.
NOCURREC	There is no current record available: the previous primitive executed on the same SCB was not successful.
NOTOPEN	The file is not open.
NOWAIT	Unauthorized simultaneous access with another process: – an attempt is made to access a record while another process is closing the file. – an attempt is made to access a record by several processes using the same SCB at the same time.
RECNFD	The record address, although it is a valid address, does not apply to an active record.
UNRECIO	An unrecoverable I/O error occurred while accessing the CI containing the requested record.
WRONGPMD	The file is opened in INPUT, OUTPUT or APPEND processing mode.

5.3.2 Using Packed-Decimal Data Type in NCL

The ability to declare and use the packed-decimal data type in NCL programs has been added to the NatStar COBOL code generator. Support for packed decimal enhances the ability of NatStar to generate COBOL code from NCL code, and enhances your ability to write NCL to access packed-decimal data in GCOS 7.

IMPORTANT:
NatStar 2.15 contains a limited release of packed-decimal support.

NOTES:

1. The NCL packed-decimal data type can be used only in programs that will be generated in COBOL.
2. NCL programs that include the packed-decimal data type cannot be tested interpretively in NCL in NatStar. These programs can be tested only by generating COBOL code, moving the code to GCOS 7 for compiling and linking, then testing in the client/server environment.

The syntax of the packed-decimal declaration in NCL is:

```
NUM VariableName (integer-part size, decimal-part size)
```

where:

`VariableName`	the name of the declared variable.
`integer-part size`	the number of digits in the integer part of the number.
`decimal-part size`	the number of digits in the decimal part of the number.

A packed-decimal constant is declared in NCL by appending a 'P' suffix to the number, as in the following examples:

```
14.21P   888.2P
```

The maximum values that can be expressed in the packed-decimal data type in NCL are the same as in COBOL. See the *COBOL-85 Reference Manual*.

EXAMPLE:

The following example illustrates a use of the packed-decimal data type in NCL.

```
LOCAL  I#(2,4)
LOCAL  NUM IP (2,3)
LOCAL  NUM LARGEIP (16,2)

MOVE 14.21P TO I#
MOVE I# TO IP
MOVE IP TO LARGEIP
IP = LARGEIP +I#
```

NCL data entities that are declared as packed decimal are generated as COMP COBOL data entities by the NatStar COBOL generator. The default form of COMP, with a trailing sign, is generated.

❑

NOTE:

COMP and COMP-3 have the same meaning for GCOS 7 COBOL.

Packed-Decimal Conversions

COBOL provides automatic conversions through the move instruction between data types when a data item of one data type is moved to another data item of a different data type. This means that data conversions for items declared as packed-decimal data type in NCL are performed automatically either by the NatStar COBOL code generator or by the GCOS 7 COBOL-85 compiler.

Compatibility of COBOL-85 and NCL Packed-Decimal Data Types

The only COBOL-85 packed-decimal data type that can be used interchangeably with the NCL packed-decimal data type is COMP. The other COBOL-85 packed-decimal data types (COMP-5, and COMP-8) either do not align the same as the NCL packed-decimal data entities, or the signs ('+' or '-') would be interpreted incorrectly.

The COBOL-85 COMP data type is compatible with NCL packed-decimal data entities. COBOL-85 COMP data entities that are declared as being unsigned can be used with NCL packed-decimal data entities with the /CBLUNSIGNED option used in the GCOS 7 Generator.

When the form of the COBOL-85 data does not map to NCL packed-decimal data, you may need to use the `NSCBL_CONVTOF` conversion routine to convert data from GCOS 7 databases so that it can be moved to and used on a client system. Refer to Section 5.3.3.6 *Converting Data Types Using NCL Data-Conversion Routines* later in this chapter for a description of how to use `NSCBL_CONVTOF.`

5.3.3 Handling Data-Type Incompatibilities

Code created in NCL supports a different -- and more limited -- set of data types than the set supported in COBOL on GCOS 7. In addition, some data types used in NCL do not map directly to COBOL data types. You need be aware of, and manage, these data-type incompatibilities in NCL programs that will be code generated in COBOL-85 for execution on GCOS 7.

IMPORTANT:

A NatStar TPR can use only NCL-compatible data types, because the logic for the TPR is defined using NCL. However, a NatStar TPR may call a conventional subroutine that uses data types that are not NCL compatible. In this case, data-type incompatibilities exist and must be handled as explained in this section.

5.3.3.1 NCL Data Types that Map to COBOL Data Types

Table 5-1 lists the NCL data types that map to COBOL data types. Only these data types can be used in NatStar TPRs for which code is generated in COBOL-85.

Table 5-1. NCL Data Types that Map to COBOL Data Types

NCL Data Type	COBOL Data Type
NUM (integer-part size, decimal-part size)	COMP signed
INT(2)	COMP-1
INT(4)	COMP-2
INTEGER	COMP-2
NUM(4)	COMP-9
NUM(8)	COMP-10
CHAR	PIC x(n)

Using NCL SEGMENTS

NCL segments, which are NCL language structures, can be mapped to COBOL structures. However, when you use NCL SEGMENTS in a NatStar TPR, you can use only data types that map directly to COBOL data types, as shown in Table 5-1.

Avoiding String Data Types in NCL

The NCL data types STRING (Pascal-style string data) and CSTRING (C-style string data) do not map directly to COBOL data types. Instead of using these data types, use the CHAR data type.

They can, however, be used if they are not converted to COBOL data types. See the example in Appendix G *NCL Sequence Listings* and the associated screen in Figure B-7 *Enter NCL for the SRV_CONC Instruction* in Appendix B.

5.3.3.2 Causes of Data-Type Incompatibility

Data-type incompatibilities can occur whenever a NatStar TPR calls a conventional subroutine. Conventional subroutines developed in COBOL can use the full set of COBOL data types, while NatStar TPRs can use only the subset listed in Table 5-1.

When developing a NatStar TPR that will call a conventional subroutine, you need to evaluate potential data-type incompatibilities in both of these situations:

- When a NatStar TPR calls a conventional subroutine, the subroutine may expect to receive values defined as data types that are not NCL compatible. The NatStar TPR cannot provide values in these data types.
- When a conventional subroutine returns values to a NatStar TPR, the values may be defined as data types that the NatStar TPR cannot handle.

There are two ways to handle these data-type incompatibilities:

1. Modify the conventional subroutine to use only NCL-compatible data types. If practical, this is the simplest method of avoiding data-type incompatibilities.
2. Convert the values between COBOL data types and NCL-compatible data types, as shown in Table 5-2, using one of the methods described later in this section.

Table 5-2. Converting COBOL Data Types Not Supported in NCL

COBOL Data Types Not Supported in NCL	NCL-Supported COBOL Data Types to Use Instead
COMP-5	COMP
COMP-8	COMP
Numeric DISPLAY (PIC [S]9(n))	COMP
GBCD	PIC X(n)

Other COBOL data types, BINARY, BIT, COMP-15, and INDEX, are not supported directly.

The two cases in which data-type conversion may be required are discussed below:

1. Passing values from a NatStar TPR to a conventional subroutine.
2. Returning values from a conventional subroutine to a NatStar TPR.

5.3.3.3 Passing Values to a Conventional Subroutine

Data-type incompatibilities can arise when a NatStar TPR calls a conventional subroutine, and values are passed as parameters on the call.

In this situation, the conventional subroutine may be coded to expect values expressed as data types that are not supported by NCL.

You have two choices for handling these situations:

1. Modify the conventional subroutine to use only NCL-compatible data types.
2. Create an "intermediary" subroutine that accepts values from the NatStar TPR, converts them to the data types expected by the conventional subroutine using COBOL MOVE statements, then calls the conventional subroutine.

   Using COBOL MOVE statements to convert data is discussed later, in Section 5.3.3.5 *Converting Data Types Using COBOL Move Statements*.

5.3.3.4 Returning Values from a Conventional Subroutine

You can convert incompatible data types to NCL-compatible data types using either of two methods:

1. Convert the data to NCL-compatible data types in the conventional subroutine as described later in Section 5.3.3.5 *Converting Data Types Using COBOL Move Statements*.
2. Convert the data to NCL-compatible data types in the NatStar TPR, using special instructions callable from NCL, as described later in Section 5.3.3.6 *Converting Data Using Types NCL Data-Conversion Routines.* Each of these methods is explained in the following subsections.

5.3.3.5 Converting Data Types Using COBOL Move Statements

You can use the COBOL MOVE statement to convert from data types that are not supported in NCL to NCL-compatible data types, as shown in the following examples.

EXAMPLES OF CONVERTING TO NCL-COMPATIBLE DATA TYPES:

```
; moving COBOL data types to NCL num (8) data type
MOVE    COMP-3 data entity      TO  COMP-10 data entity.
MOVE    COMP-5 data entity      TO  COMP-10 data entity.
MOVE    COMP-8 data entity      TO  COMP-10 data entity.
MOVE    PIC[S]9(n) data entity  TO  COMP-10 data entity.

; moving COBOL data types to NCL CHAR (x) data type
MOVE    GBCD data entity        TO  PIC X(n) data entity.
```

The above examples show data conversion from COBOL data types not supported to NCL-compatible data types. Statements similar to these examples can be used in a conventional subroutine to convert data before returning control to a NatStar TPR.

❑

You can also use COBOL moves to convert from NCL-compatible data types to other COBOL data types; to do so, simply reverse the moves in the above examples. This type of conversion is required in an "intermediary" routine that accepts NCL-compatible data types from a NatStar TPR and converts them to data types that can be handled by a called subroutine.

5.3.3.6 Converting Data Types Using NCL Data-Conversion Routines

The preceding section described how to convert data types using COBOL MOVE statements. This section describes another method, which is to call an NCL data-conversion routines when you are developing the NatStar TPR. In this case, the NCL-incompatible data types are returned from a conventional subroutine to the NatStar TPR, and are converted there.

The NCL data-conversion routines described in this section can convert only COMP, numeric DISPLAY, and GBCD data types. Other data types can be converted only by using COBOL MOVE statements.

IMPORTANT:
Depending on the data types and values concerned, some precision could be lost.

Functions

You can call either of two data-conversion routines by including a special instruction in your NCL program. These special instructions are:

`NSCBL_CONVTOF`	Convert the passed data entity to NUM(8); i.e., double-precision floating-point data type
`NSCBL_CONVTOC`	Convert the passed data entity to CHAR data type

The syntax to call these special instructions is as follows:

Syntax

```
NSCBL_CONVxxx ret_status, ptr_in_val, output_val, type,
                  length, scale
```

Syntax Rules

NOTE:
All of the arguments are defined as in NCL as input/output parameters.

1. `NSCBL_CONVxxx`

   The name of the specific data-conversion routine to be called (`xxx = TOF` or `TOC`).

2. `ret_status`

   The status value returned by the data-conversion routine. The possible return status values are shown in Table 5-3. This parameter must be defined in NCL as data type INT(4).

Table 5-3. Status Values Returned by the NCL Data-Conversion Routines

Value	Description of the Error
0	No error occurred
-2	Conversion not supported (e.g., GBCD to floating point)
-3	Value was truncated
-4	Overflow occurred during conversion
-5	Underflow occurred during conversion
-6	Invalid digit/sign
-8	Error in parameters when routine was called

3. `ptr_in_val`

   Pointer to the data entity to be converted. This parameter must be defined in NCL as data type POINTER.

4. `output_val`

   The converted data entity returned by the routine. The `output_val` parameter must be defined as the data type to which the `in_val` is being converted; i.e., either CHAR or NUM(8). For items being converted to CHAR, the `output_val` definition must be large enough to contain the converted data item.

5. `type`

   The data-type code of the data entity being passed to the routine. Data-type codes are listed in the following table. This parameter must be defined in NCL as data type INT(4).

Table 5-4. Data-Type Codes Accepted by NCL Data-Conversion Routines

Name	Data Type	Description
Packed decimal, leading sign	1	Scaled or unscaled, packed decimal, leading sign. COBOL COMP-3 SIGN IS LEADING
Decimal, leading sign	2	Scaled or unscaled, unpacked decimal, leading sign. COBOL PIC S9(n) SIGN IS LEADING SEPARATE
Decimal, trailing sign	11	Scaled or unscaled, unpacked decimal, trailing sign. COBOL PIC S9(n) SIGN IS TRAILING SEPARATE
Decimal, no sign	12	Scaled or unscaled, unpacked decimal, no sign. COBOL PIC 9(n)
Packed decimal, trailing sign	14	Scaled or unscaled, packed decimal, trailing sign. COBOL PIC S9(n) COMP-3 SIGN IS TRAILING
Packed decimal, no sign	15	Scaled or unscaled, packed decimal, no sign. COBOL PIC 999V99 COMP-3
Binary-Coded Decimal (BCD)	18	BCD characters, each occupying 6 bits. COBOL PIC X(20) GBCD **NOTE:** GBCD can only be converted to CHAR; i.e., NSCBL_CONVTOF cannot be used on this data type.

6. `length`

The length of the data entity to be converted. For numeric values, the `length` value represents the number of digits plus the sign of the data to be converted. For CHAR entities, the `length` value is the size of the field. This parameter must be defined in NCL as data type INT(4). Some examples of how to use `length` are shown below:

```
COBOL data definition        LENGTH value

PIC S99v99                          5
PIC 99v99                           4
PIC S999v9999 COMP                  8
PIC 9999v9999 COMP                  8
PIC S9(7) COMP                      8
PIC 9(8) COMP                       8
```

7. `scale`

The scale factor to be applied to the decimal or packed-decimal number to be converted. Required only when the data entity is a numeric data type; for CHAR data type, set the `scale` to zero. The scale factor represents the location of the decimal point. This parameter must be defined in NCL as data type INT(4). If no decimal point is present, the `scale` value is zero.

A negative number `scale` indicates the number of digits to the right of the decimal point. A positive `scale` multiplies the number by a power of 10, with the `scale` value representing the exponent value.

Some examples of how to use `scale` are shown below:

COBOL data definition	**SCALE value**
PIC S99v99	-2
PIC S999v9999 COMP	-4
PIC S9(7) COMP	0

Example of NSCBL_CONVTOF use:

The following example illustrates how to call the `NSCBL_CONVTOF` instruction from an NCL program.

Segment Definitions for Data Items to be Converted

Assume that your NCL program includes the following segment definitions, and that you need to convert the two fields named `DEC_VAL_1` and `PD_VAL_1`.

```
SEGMENT DEC_DATA_AREA
      CHAR TXT_BIG_NAME (40)
      CHAR TXT_SMALL_NAME (25)
      CHAR TXT_PHONE (11)
      CHAR DEC_VAL_1 (20)       ; decimal item to be converted
      CHAR DEC_VAL_2 (20)
ENDSEGMENT
;
SEGMENT PD_DATA_AREA
      CHAR TXT_ADDRESS (40)
      CHAR TXT_NAME (20)
      CHAR PD_VAL_1 (20)        ; packed-decimal item to be
                                ; converted
      CHAR PD_VAL_2 (20)
ENDSEGMENT
```

Constant Definitions

```
; Constants used to identify data type of item to be converted
; (data-type codes were defined earlier, in Table 5-4)
;
CONST PACDEC_TS% 14         constant for type packed-decimal, trailing sign
CONST DECIMAL_NS% 12        constant for type decimal, no sign
```

Instruction Definition

The following instruction is part of a subroutine whose arguments are FLOAT_VAL_1 and FLOAT_VAL_2.

```
; Local data variables
;
LOCAL NUM OUT_FLOAT_1#(8)
LOCAL NUM OUT_FLOAT_2#(8)
;
LOCAL PD_DATA_AREA X
LOCAL DEC_DATA_AREA Y
;
; Return status variable
;
LOCAL INT RET_STATUS(4)
; Input variables passed to NSCBL_CONVTOF
;
LOCAL INT IN_DATA_TYPE(4)
LOCAL INT IN_DATA_LENGTH(4)
LOCAL INT IN_DATA_SCALE(4)
LOCAL POINTER DATA1_PTR
LOCAL POINTER DATA2_PTR
;
; Prepare the input variables to convert a signed packed-
; decimal field with trailing sign [COBOL PIC S9(5)V99 COMP]
; to floating point.
;
MOVE PACDEC_TS% TO IN_DATA_TYPE       ; Set the data type
MOVE -2 TO IN_DATA_SCALE              ; Set the scale
MOVE 8 TO IN_DATA_LENGTH              ; Set the data length
                                      ; to number of digits
                                      ; plus the sign
;
; Call NSCBL_CONVTOF to perform the conversion from decimal
; to floating point
;
MOVE @X.PD_VAL_1 TO DATA1_PTR
NSCBL_CONVTOF RET_STATUS, DATA1_PTR, OUT_FLOAT_1#,        \
      IN_DATA_TYPE, IN_DATA_LENGTH, IN_DATA_SCALE
;
; Check the return status
;
IF RET_STATUS <> 0                    ; Non-zero = error
      MOVE -1 TO OUT_FLOAT_1#
      EXIT
ENDIF
MOVE OUT_FLOAT_1# TO FLOAT_VAL_1
;
```

```
; Prepare the input variables to convert an unsigned decimal
; field [COBOL PIC 9(5)V9] to floating point.
;
MOVE DECIMAL_NS% TO IN_DATA_TYPE      ; Set the data type
MOVE -1 TO IN_DATA_SCALE              ; Set the scale
MOVE 6 TO IN_DATA_LENGTH              ; Set the data length
                                      ; to number of digits
                                      ; plus the sign
;
; Call NSCBL_CONVTOF to perform the conversion from decimal
; to floating point
;
MOVE @Y.DEC_VAL_1 TO DATA2_PTR
NSCBL_CONVTOF RET_STATUS, DATA2_PTR, OUT_FLOAT_2#,        \
      IN_DATA_TYPE, IN_DATA_LENGTH, IN_DATA_SCALE
;
; Check the return status
;
IF RET_STATUS <> 0                    ; Non-zero = error
      MOVE -1 TO OUT_FLOAT_2#
      EXIT
ENDIF
MOVE OUT_FLOAT_2# TO FLOAT_VAL_2
```

5.3.4 Using Native Code

IMPORTANT:
The advantage of coding in NCL is that NCL is platform independent. Logic developed in NCL can be ported to any platform supported by NatStar. Using native code can severely limit portability.

When developing a NatStar TPR (or other code to be used on GCOS 7), you can use the native-code capability to include COBOL-85 code in the NCL source. Native code is sometimes called inline code.

NOTE:
The use of native code restricts the target source to a single language.

You can share variables between native code and NCL code by using the naming rules for variables shown in the following table.

Table 5-5. Mapping NCL Variable Names to Names Used in Native Code

NCL	COBOL-85
Var$	SZ-VAR
Var%	I-VAR
Var#	R-VAR

The $ suffix defines a character variable; the % suffix defines an integer variable; the # suffix defines a real-number variable.

If the NCL variable is defined without a suffix using a GLOBAL or LOCAL statement, its full name is retained without a prefix. For example:

`LOCAL INT VAR (4)` maps to `VAR` in COBOL

EXAMPLE:

The following example illustrates the use of native code. (This example is for illustration purposes only; the same result could have been obtained by coding in NCL.) The purpose of the example application is to enter an employee ID and obtain the employee's name and salary. The client application presents the user with a window in which the user enters employee ID, then clicks on the push button FIND.

The application either:

a. returns the employee name and salary; or

b. returns a diagnostic window if no employee is found with the employee ID entered by the user.

In this example, the NCL code belongs to the function `GET_EMPLOYEE_INFO%`. This is a remote function, which is to be executed on GCOS 7. The function has one input parameter (`PASS_EMP_ID`) and two output parameters (`PASS_EMP_NAME` and `PASS_EMP_SALARY`). The function also returns an INTEGER (4) value that is used to verify that the function executed correctly. For brevity, the input/output parameters defined in NCL are not shown.

```
LOCAL EMPLOYEE_NAME$, SALARY#, EMPLOYEE_ID%, RETURN_CODE%

; Move the passed value in the PASS_EMP_ID parameters of
; the function into the variable EMPLOYEE_ID%

MOVE PASS_EMP_ID TO EMPLOYEE_ID%

; Inclusion of native COBOL-85 Call User Subroutine source,
; starting at position 8 because the GCOS 7 COBOL compiler
;  is position-sensitive
```

```
; Note that the variable names passed in the CALL statement
; have been derived as described in Table 5-5.

%
       MOVE I-EMPLOYEE-ID TO EMP-NO.

       CALL "FIND-EMPLOYEE" USING I-EMPLOYEE-ID,
                                  SZ-EMPLOYEE-NAME,
                                  R-SALARY,
                                  I-RETURN-CODE.
%
; End of inclusion of native COBOL-85 Call User Subroutine
; source

; If an Employee has been found, move the value in variable
; EMPLOYEE_NAME$ to the PASS_EMP_NAME output parameter
; and the value in variable SALARY# to the PASS_EMP_SALARY
; output parameter. Return a value of 0 from the function.
; If no Employee has been found, return a value of 1 from
; the function.

IF RETURN_CODE% = 0
       MOVE EMPLOYEE_NAME$ TO PASS_EMP_NAME
       MOVE SALARY# TO PASS_EMP_SALARY
       RETURN (0)
ELSE
       RETURN (1)
ENDIF
```

❑

Native code entered in an NCL program is copied verbatim, minus the beginning and ending % (as shown in the preceding example), into the output source code when the NCL program is processed by the NatStar code generator.

5.3.5 Calling A Conventional Subroutine

This section explains how to call a conventional subroutine from a NatStar TPR. Conventional subroutines do not use NatStar Communication Services (NSCOM7) and typically are developed using conventional methods (e.g., by programming in COBOL-85).

IMPORTANT:
When you call a conventional subroutine from a NatStar TPR, you need to be concerned about data-type incompatibilities. Refer to Section 5-47 *Handling Data-Type Incompatibilities* earlier in this chapter.

You can call a conventional subroutine by using a local library in NCL to describe the conventional subroutine, as described later in this section. Before doing so, you need to handle the problem, if it exists, of dashes in the name of the conventional subroutine. If the name(s) of the conventional subroutine(s) you need to call do not include dashes, go directly to *Using a Local Library to Call a Conventional Subroutine* later in this section.

Handling Dashes in the Name of the Conventional Subroutine

If the GCOS 7 name of the conventional subroutine contains a dash ("-"), you need to change the name of the subroutine. NatStar does not understand the use of a dash as a separator. NatStar allows only the underscore ("_") to be used as a separator in the name of an instruction or function.

You can use either of these two methods to change the name of a subroutine when the name includes a dash:

1. Replace the dashes with underscores when referring to the subroutine in NCL. For example, in NCL use RETRIEVE_ALL_ACCOUNTS when referring to a subroutine whose name on GCOS 7 is RETRIEVE-ALL-ACCOUNTS. Then in the Link Edit activity that builds the GCOS 7 library that includes the conventional subroutine, use the REPLACE option to rename the subroutine that includes underscores (RETRIEVE_ALL_ACCOUNTS) to the name that includes dashes (RETRIEVE-ALL-ACCOUNTS). Refer to the *GCOS 7 Linker Reference Manual* for more information about using REPLACE.
2. Use the EXTERNAL field when defining an instruction or function in NCL. In this field, enter the name of the conventional subroutine, using the dash(es). If you use this method, you will need to both:

   a) create a local library for the conventional subroutine, as described below; and

   b) generate a Dynamic Link Library (DLL) on the workstation. Refer to the *NatStar Getting Started* manual for information about how to generate a DLL. (Order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide.)

Using a Local Library to Call a Conventional Subroutine

Define a library that includes each subroutine, defined as an instruction with its respective parameters. Do not generate COBOL code for this library. The library is used only to resolve the unresolved references that would be generated (resulting in an "invalid data name" error message) if the instructions were included in an NCL statement.

To reference any instructions in this library, the library must be Selected as "usable" from the library that calls the procedures in the local library. To do so, be sure that the library is listed in the **Libraries used by...** window for the library from which it is called. Appendix C, Step 2.3, illustrates this process.

Using Native Code to Call a Conventional Subroutine

Another way to call a conventional subroutine that exists on GCOS 7 is to use native code in the NCL program. Refer to Section 5.3.4 *Using Native Code* earlier in this chapter for an explanation of how to use this feature. As noted in that discussion, the use of native code limits program portability, and should in general be avoided.

IMPORTANT:
In all cases a conventional existing subroutine must return to the calling routine and must not execute any SEND/RECEIVE statements or NSCOM instructions (except access to a file or database, or sending message to a slave terminal, i.e. non-NatStar client).

5.3.5.1 Restrictions on Message Receive/Send Operations

Do not include message receive or send operations from/to a client in a NatStar TPR. NSCOM performs the operations necessary to receive input transaction messages and to send output transaction messages, as explained in Section 5.1.2 *NatStar TPR Structure and Processing Flow* earlier in this chapter.

NOTE:
The only exception to the above restriction is SENDing a message to an explicit destination. Sending to a slave terminal is permitted in a NatStar TPR. A NatStar client cannot be an explicit destination.

Messages

Use of the following instructions is restricted:

Send instructions cannot be used to send a message to any NatStar client. If you need to use a broadcast function at the TDS, be sure to exclude NatStar clients so that the message is not transmitted to them. To avoid this situation, you may want to configure a TDS used only for NatStar TPRs and clients (refer to Section 3.2 *Configuring TDSs* in Chapter 3 for additional discussion of this topic).

If a message is erroneously addressed to a NatStar client, an error is sent to the client addressed by the message. Refer to Section 5.3.6 *Handling Errors* later in this chapter for additional information about errors and how to handle them.

5.3.5.2 Other Restrictions

NSCOM does not support conversational transactions. Therefore, do not use the receive instruction in native COBOL source.

The NEXT TPR mechanism is not supported.

5.3.6 Handling Errors

In addition to any errors that may be inadvertently coded in the application logic, several kinds of system-detected errors can occur during the processing of a NatStar TPR. These errors can be categorized as follows:

1. Errors that can occur during the processing of any TPR, including processing against UFAS files.

   Some of these errors are listed at the end of this paragraph.

2. Errors associated with the TCP/IP connection between the client and the NatStar TPR on GCOS 7, and errors detected by the NatStar Communication Services software. Refer to the *NatStar Communication Service User and Programming Guide*.

Errors in all two categories will result in an error status being returned to the client application at the point where it attempted to execute the server instruction that failed because of an error.

You can debug errors in Category 1 by using traditional TDS debugging methods (e.g., by analyzing the TPR dump, etc.). This section provides additional information about errors in Category 2.

IMPORTANT:
You should use one of the techniques described below to detect and report errors. If you do not use one of these techniques, these errors will be undetected and unreported, and the results returned by the failed server instruction may be unpredictable.

Error handling can be divided into two phases:

1. Error detection and reporting
2. Error analysis and resolution

5.3.6.1 Detecting and Reporting Server and Communication Errors

Errors that occur during client/server application execution are returned to the client application. You have two choices for detecting and reporting these errors:

- **Automatic error detection and reporting**. If you choose this mode, NatStar automatically detects any errors and displays a dialog box (to the user) that describes the error. This choice is convenient, because no NCL coding is required, but does not provide you with any programmatic control over error handling.
- **Programmatic error detection and reporting**. If you choose this mode, you must include error tests and error-reporting statements in the application NCL. This choice provides more control over error handling, but increases the amount of NCL coding required.

If neither choice is activated, errors are ignored and results returned by a failed instruction are unpredictable.

If you choose both automatic and programmatic error detection and reporting, the automatic mode occurs first, followed by the programmatic mode.

Both of these situations are discussed in the remainder of this section.

Automatic Error Detection and Reporting

When Automatic Error Detection and Reporting is activated, any error detected by NSCOM, including the server and communication errors listed above, causes an error popup window similar to the one shown in Figure 5-3 to be displayed.

Automatic Error Detection and Reporting can be very useful when you are debugging a client/server application. However, this mode could be confusing to a non-technical user of production Bull/NatStar applications.

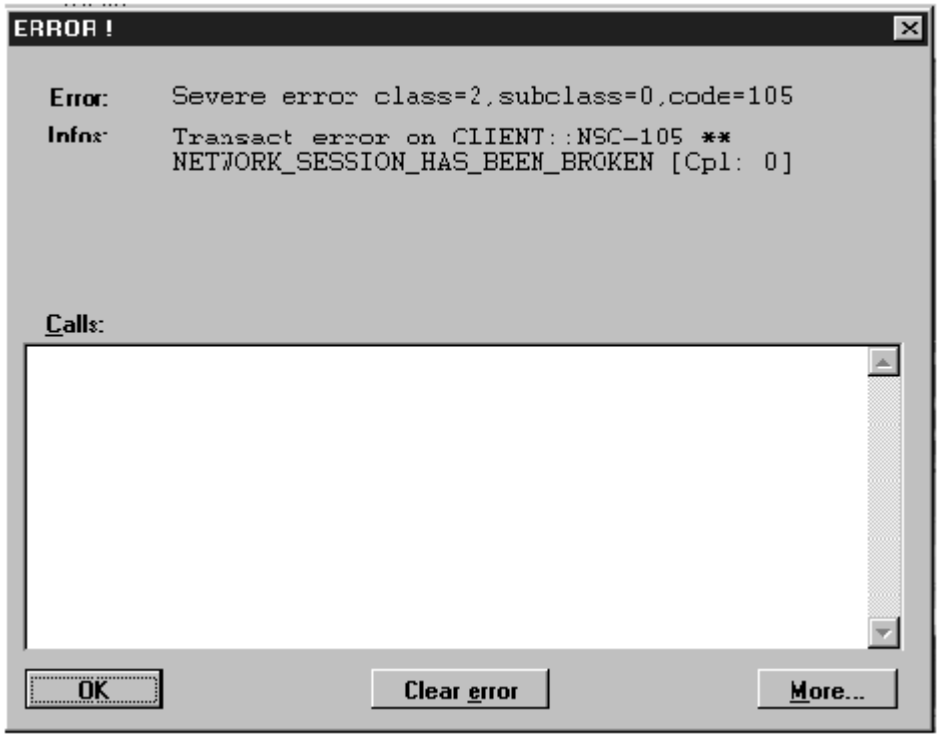

Figure 5-3. Error Popup Window

When the Error Popup window is presented, you can click on **OK**. If the error is fatal, the client application is aborted. If the error is not fatal, application execution continues with the next NCL statement (refer to the later discussion of *Programmatic Error Detection and Reporting*).

The use of Automatic Error Detection and Reporting features is controlled by the Configuration you choose when building the client application.

To activate Automatic Error Detection and Reporting for a Configuration:

1. First select the **Configuration...** entry from the **Options** menu.
2. Then, in the **Select Default Configuration** dialog box, select the name of the desired client Configuration and click on the **Misc...** button.
3. In the **Miscellaneous** dialog box, select the **IM** tab and click on the **Error popup** checkbox so that it is activated -- i.e., displays a check mark.
4. Close both dialog boxes by clicking on their **OK** and **Select** buttons.

The above procedure is illustrated in the example application in the appendices.

Programmatic Error Detection and Reporting

If you do not plan to use Automatic Error Detection and Reporting in the production version of the application, you should use the NCL statements described below to check for errors at appropriate points in the program. A detailed discussion of exactly when to check for errors (and how to report them) is beyond the scope of this document. However, good practice is for the client logic to check for an error after calling any instruction that executes as a remote service.

Understanding how NSCOM processes errors is useful when deciding how to use NCL to check for an error. When an error is detected, NSCOM:

- Displays the Error Popup Window (refer to Figure 5-3) if the Error Popup checkbox was activated.
- Stores information about the error, so that the NCL developer can access the information as described below.
- If the error was not fatal, resumes execution of the application program with the next NCL statement (which may or may not be an error test).

The stored error information is typically retained until it is overwritten by information about another error, or until it is cleared explicitly using an NCL instruction as described below.

Because the stored error information can persist over the execution of many NCL statements, error detection and reporting could be deferred; for example, until the end of a routine. However, if multiple errors occurred in the routine, only the most recent error could be detected. (The Error Popup window, if activated, would display each error as it occurred.)

Because typically the stored error information is not cleared automatically, an error that occurred in one set of NCL (e.g., the EXECUTED event of a button) can be detected by another set of NCL. If this situation is unintentional, the result is likely to be confusing. To avoid this problem, you can use a sequence of NCL similar to the following:

1. Execute an NCL instruction to clear any stored error information.
2. Execute the NCL statement(s), such as a call to a remote server instruction, that will be error-checked.
3. Test for an error in the preceding set of NCL statements. Only the most recent error can be detected if multiple errors occurred.
4. Report the error in a way that is consistent with the application design. For example, log the error and/or display a dialog box to the user.

You can use the following NCL statements to erase the stored error record, detect that an error occurred, and provide information to be used when reporting errors.

- **Clearing error storage.** You can use the following NCL instruction to clear any error information stored by the NatStar runtime routines:

  ```
  THINGS_CLEAR_ERROR
  ```

  The `THINGS_CLEAR_ERROR` instruction does not require any arguments. This instruction is described in the NatStar *Programming Interfaces Libraries Volume 3* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

- **Checking for an error.** You can use the following NCL function to check for an error detected by the NatStar runtime routines during the execution of a prior NCL statement (note that this is an NCL **function**, not instruction):

  ```
  THINGS_GET_ERROR%
  ```

  The `THINGS_GET_ERROR%` typically is used in an NCL conditional (e.g., `IF`) statement. This function does not require any arguments, and returns the NatStar error number if an error occurred, or a zero if there was no error. The use of this function is illustrated in the example application in the appendices.

IMPORTANT:
Do not confuse the `THINGS_GET_ERROR%` **function**, whose name ends with a percent sign, with the `THINGS_GET_ERROR` **instruction**, which has no percent sign, described below.

- **Reporting an error.** When you have determined that an error occurred, you can use the following NCL function to obtain a description of the error (note that this is an NCL **function**, not instruction):

  ```
  THINGS_GET_ERRMSG$
  ```

  The `THINGS_GET_ERRMSG$` function does not require any arguments, and returns a string of text that describes the error. The use of this function is illustrated in the example application in the appendices.

  You can obtain additional information about the error by using either of the following NCL **instructions** (not functions):

  ```
  THINGS_GET_ERROR  <arg1> ... <arg5>
  THINGS_GET_DIAGNOSTIC  <arg1> ... <arg6>
  ```

  Each of these instructions returns a series of fields that contain information to describe the error. Each field is returned in one of the instruction arguments.

These two instructions are almost identical, except that `THINGS_GET_DIAGNOSTIC` returns one additional argument (the sixth). Both of these instructions are described in the NatStar *Programming Interfaces Libraries Volume 3* and the *NatStar Development Guide, Information Modeling* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

Suggested Error-Handling Technique

As the above discussion illustrates, the requirement to clear errors, then invoke a remote procedure, then test for errors can distract application developers from the central task of defining the logic for business solutions.

These requirements further complicate the process of partitioning client/server logic, because developers must add error-processing logic to existing code if a routine that was local is moved to a remote library.

To avoid these complications, a general technique for error-handling such as the following is very useful.

Always invoke a remote procedure indirectly, using site-developed interface routines that perform the necessary tests.

For example, instead of coding:

```
THINGS_CLEAR_ERROR
MY_GET_EMPLOYEE result%, . . .
IF THINGS_GET_ERROR% THEN
 ; do required error processing
ENDIF
```

Develop an interface routine that encapsulates these functions. Instead of the above, you can simply call:

```
GET_EMPLOYEE result%, . . .
```

5.3.6.2 Error Analysis and Resolution

The process of detecting and reporting an error typically is followed by analysis and resolution, which determine how to handle the error. The following discussion treats error analysis and resolution in terms of three options:

1. Use the information from the Error Popup and/or NCL instructions described earlier in Section 5.3.6.1 *Detecting and Reporting Server and Communication Errors*.
2. For client-related problems, obtain additional information from the NatStar Debugger and/or NatStar client trace files.
3. For server-related or communication problems, obtain additional information from the communication traces of NatStar.

Analyzing Error Code and Message

The error code and message provided by the Error Popup and/or NCL instructions may be sufficient to analyze the error and determine how to handle it.

If this is not the case, some additional information is available in the Nat Systems documentation. In some error messages, the error code is preceded by the prefix `NST-` or `NSC-` (refer to Figure 5-3). Errors with the `NST-` or `NSC-` prefix are described in the *NatStar User Guide and Programming Manual --Communication Services* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

However, even this additional information may be insufficient to determine the cause of the error. One example is the following message:

```
NSC-105 NETWORK_SESSION_HAS_BEEN_BROKEN
```

(This message is shown in Figure 5-3). You will need additional information to determine *why* the session has been broken.

You can obtain additional information by using either of the two following methods:

- Using the NatStar Debugger and Client Trace Analysis
- Analyzing Log Files

Using the NatStar Debugger and Client Trace Analysis

If the error originated in the client, *not* in the server or the communication path to the server, NatStar provides problem-analysis tools that include a debugger (refer to the User Manual portion of the *NatStar Adaptable Development Environment Installation and User Manual*) and a NatStar trace mechanism (refer to the *NatStar Development Guide, Information Modeling*).

Analyzing the Log File with NSCOM7

If the error was detected by GCOS 7 and/or the Networking Services software that handles the connection to GCOS 7, you can obtain additional information from the NatStar log file `NSCOMTCP.LOG`.

You should analyze `NSCOMTCP.LOG`, as described here, before analyzing other trace and/or log files. If you are unable to diagnose the problem using information from `NSCOMTCP.LOG`, consult the documentation for the Networking Services software installed at the site for information about how to use its trace/logging data.

The NSCOM Log File, `NSCOMTCP.LOG`, is neither created nor written to unless activated explicitly. To activate `NSCOMTCP.LOG`, you need to add a `DIRECTORY=` entry to the `[SYSTEM]` section of the `NSCOM.INI` file. The `NSCOM.INI` file can be found in:

`c:\`*`NatStar`*`\INI\NSCOM.INI`

where:

`c:\`*`NatStar`*`\`

is the NatStar installation directory chosen when the NatStar software was installed.

For example, if the `[SYSTEM]` section of `NSCOM.INI` contains:

```
[SYSTEM]
NODE       = MACHINE1
STATS      = NO
MAXBUFSIZE = 4096
DIRECTORY  = C:\NatStar\
```

Then the NatStar NSCOM Log file can be found at:

```
c:\NatStar\LOG\NSCOMTCP.LOG
```

The **amount** of information written to the `NSCOMTCP.LOG` file is controlled by the `PROTOCOLTRACE=` parameter in the [TCPIP] section of the NSCOM.INI file. `PROTOCOLTRACE=NO` provides the information described below, but `PROTOCOLTRACE=YES` provides much more information.

When running multiple tests to attempt to analyze a problem, you may find it helpful to rename (or delete) NSCOMTCP.LOG before each test. Doing so allows you to isolate the messages generated by each new attempt, because information is typically appended to the end of the log file.

Because of the large number of potential error causes, and because the information returned varies depending on which Networking Services software is being used, listing all of the possible error messages and their causes is impractical. However, the following generalizations are based on experience:

- If the communication path to GCOS 7 is new or has been reconfigured, and if NSCOMTCP.LOG was not created, is empty, or contains only one (new) line, verify the parameters in the NSCOM.INI file. Invalid NSCOM.INI parameters frequently cause errors.
- If the communication path and NSCOM.INI file worked earlier and neither has been changed, and if a single (new) line was written to NSCOMTCP.LOG (for example, CM_ALLOCATE_FAILURE_NO_RETRY), verify that the TDS system and all nodes in the communication path are operational.
- If a relatively large amount of information was written to NSCOMTCP.LOG, the communication path and server TDS are probably operational, and the problem may have been detected by the TDS. In this case, proceed with the analysis as described in the following paragraphs.

Specific TDS errors collected by NatStar

- TDS mailbox I/O error
- TDS system error
- TDS invalid frame
- TDS user not connected
- TDS terminated

The error NST 572 allows you to collect all TDS errors which cause a message that does not represent an NSCOM frame to be sent to the mailbox.

In particular, in the case of TPR ABORT, the error message received in the mailbox will not be transmitted to the client, but put in a <TDSname>.log file in the "/usr/natsys/ini" directory.

5.4 Generating Server Source Code

After you have completed the development and interpretive debugging of the client/server logic in NCL, you need to generate source code for the server part of the application. The server part will become the procedure in a NatStar TPR. This section describes how to generate server source from the NCL.

NOTE:
To test the generated server source code, you do *not* need to generate code for the *client* part of the application, because you can continue to execute it interpretively on the NatStar development workstation.

This section first provides some background information about NatStar code generation, then describes how to generate COBOL source. For additional information about the topics discussed in this section, refer to the *NatStar Development Guide -- Client-Server* and to the example application in the appendices. The order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide.

5.4.1 Background Information

This section summarizes general information about generating code from NCL.

Using Separate Client and Server Libraries

As described earlier in this chapter in *Using the NatStar Development Workstation*, client and server logic must be packaged in separate libraries. Source code will be generated only from the server library in the procedure described here.

Changing the Workspace Settings

When NATSTAR asks you to create/change the Workspace, you set the option COBOL Generation (as shown in Figure 5-4).

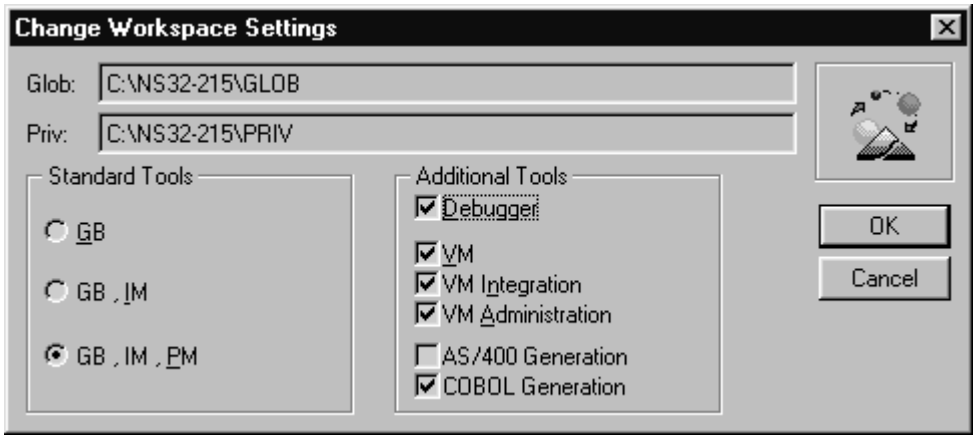

Figure 5-4. Change Workspace Settings

Creating the Targets

You must create 2 targets:

- For the Language.
- For the User Interface.

The Language Target Type

To create the Language target type:

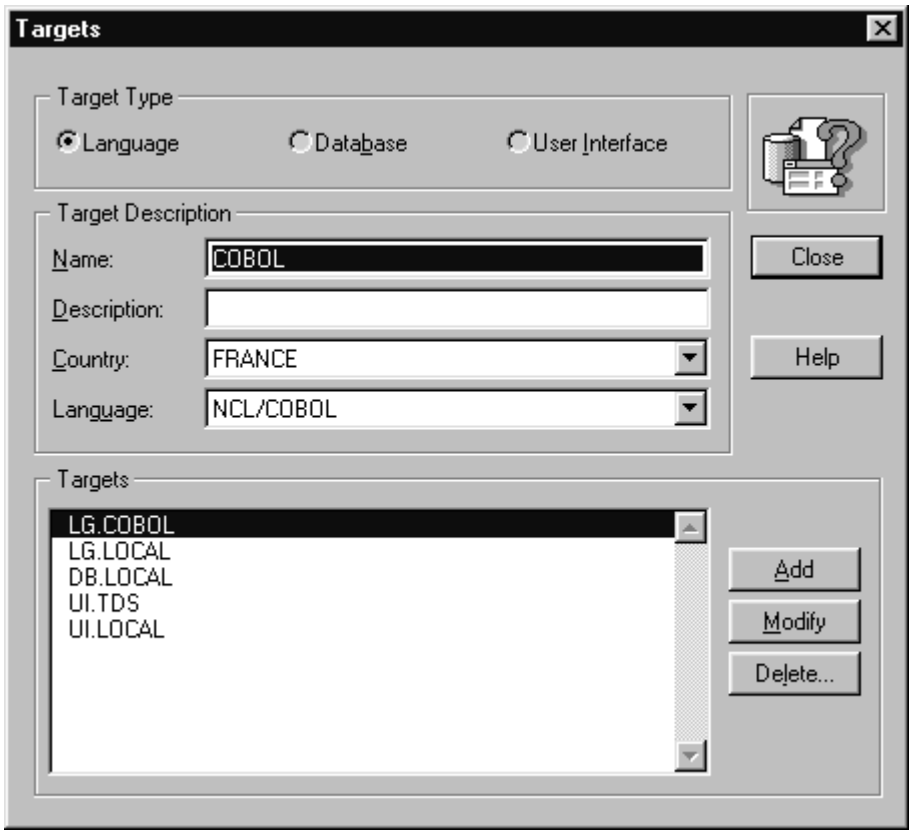

Figure 5-5. Creating the Language Target Type

The User Interface Target Type

To create the User Interface target type:

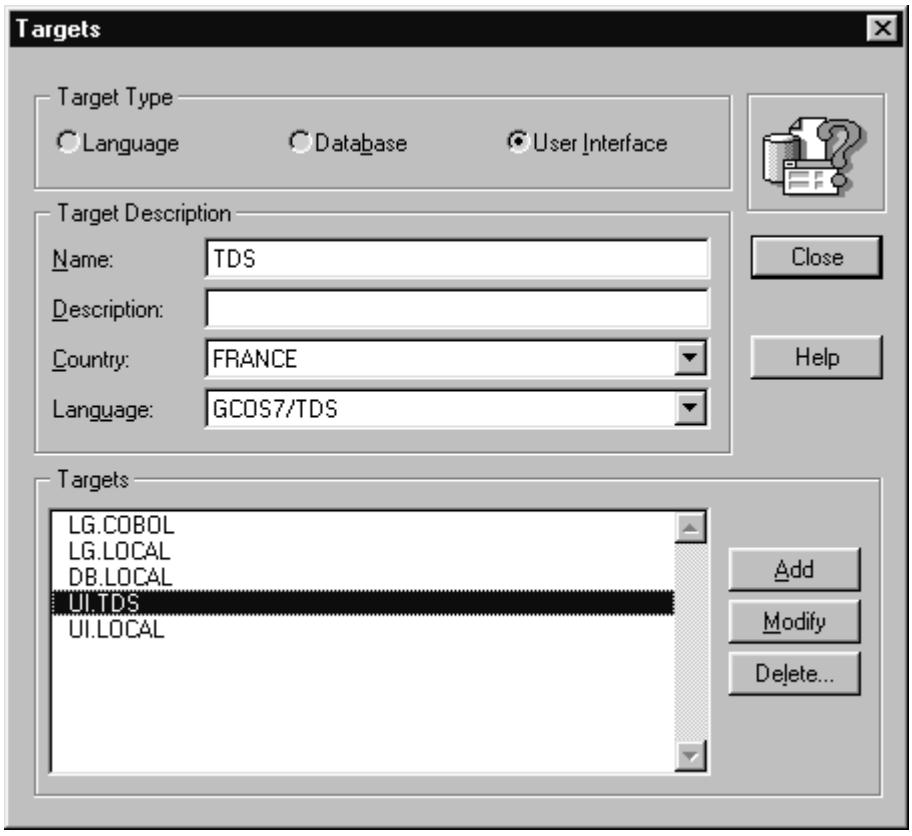

Figure 5-6. Creating the User Interface Target Type

Installing the Services

COBOL Libraries

For the COBOL Libraries, install the services (shown in the following figure) from the: %NATSTAR%\gcos\ncb directory:

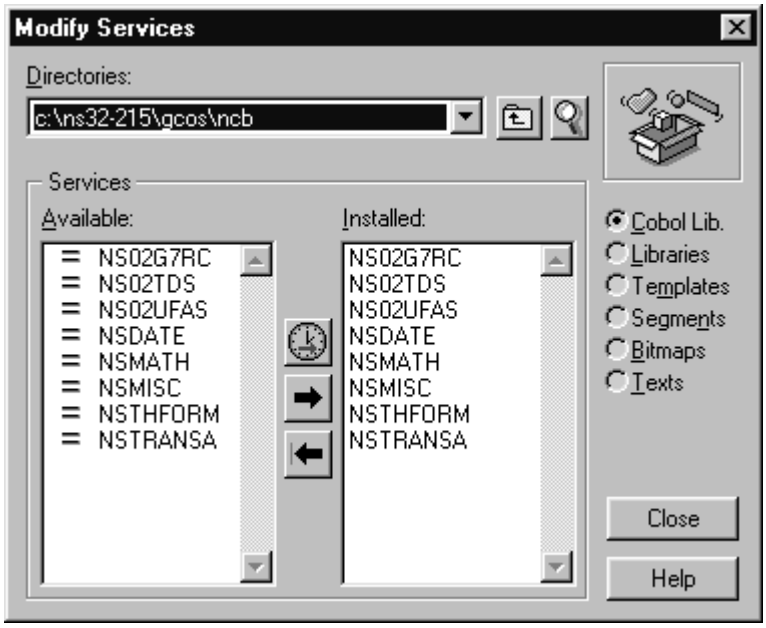

Figure 5-7. Installing Services for COBOL Libraries

NCL Libraries

For the NCL Libraries, install the services (shown in the following figure) from the: %NATSTAR%\NCL directory (the NS02OP7G is useful if NSCOM Light is used)

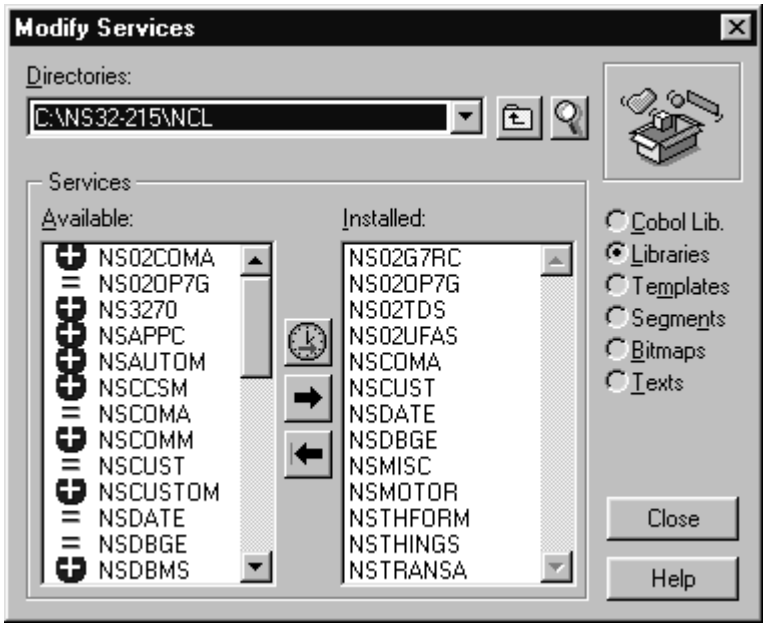

Figure 5-8. Installing Services for NCL Libraries

Miscellaneous Server Configurations

You must define a server configuration like that shown in the following figures.

You need at least 1 server configuration for GCOS 7/TDS generation. Let us say that the configuration name is GCOS7_TP.

Main Tab

For the Main configuration, choose "Server" and "Remote Generation".

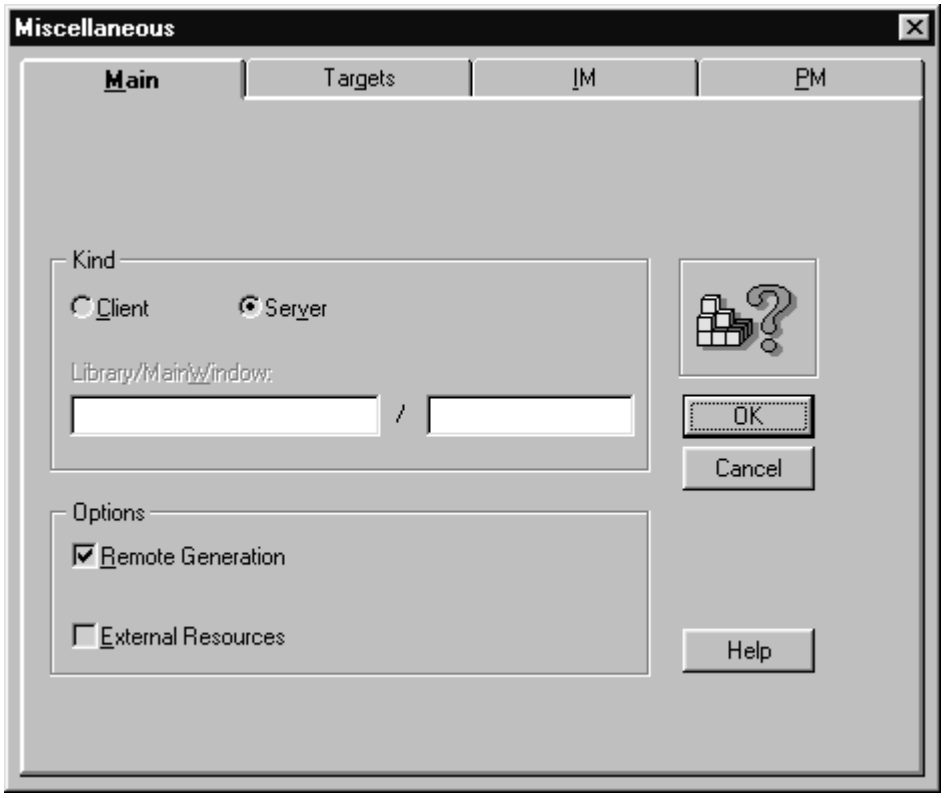

Figure 5-9. Main Configuration

Target Tab

For the Target Configuration, specify the targets already defined (above):

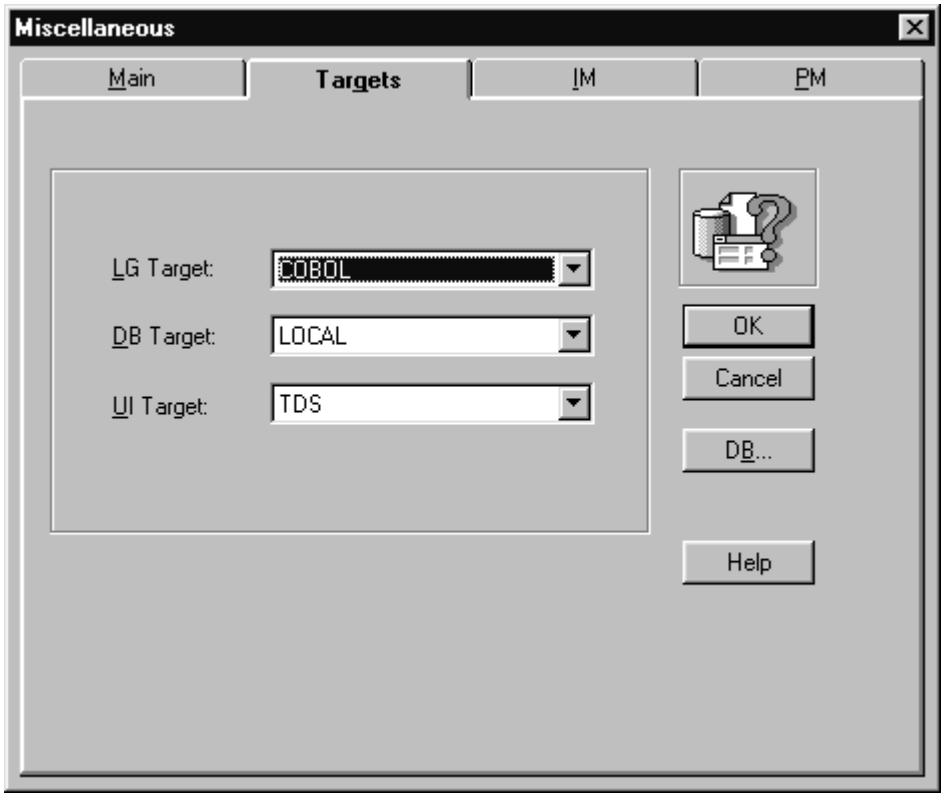

Figure 5-10. Target Configuration

IM Tab

For the IM configuration, specify the Transaction monitor, either NS02COMT (as shown in Figure 5-11) to use NSCOM, or NS02OP7G to use NSCOM Light.

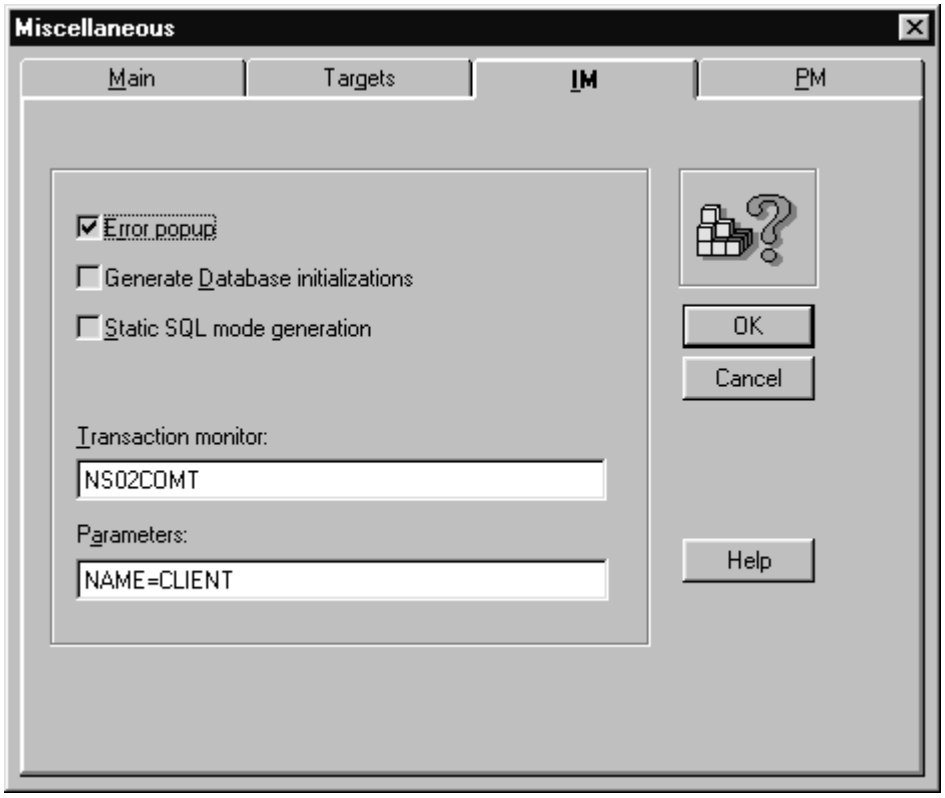

Figure 5-11. IM Configuration

PM Tab

For the PM configuration, no special characteristics are required:

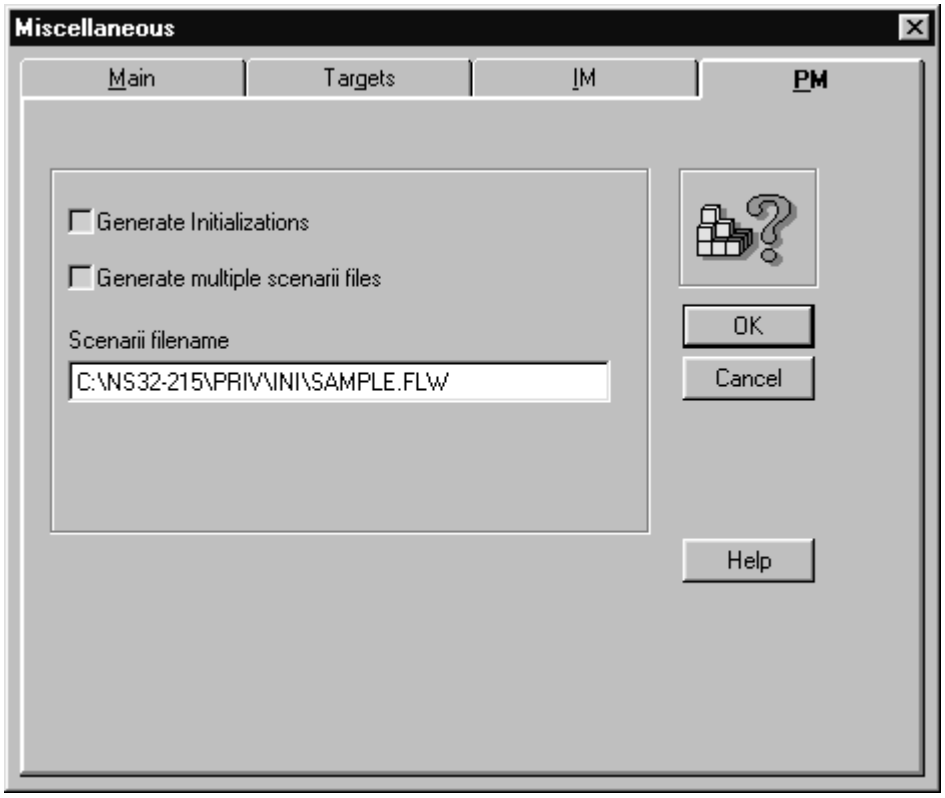

Figure 5-12. PM Configuration

Server Configuration/Generation

Main Tab

The **Main** Tab allows you to transfer automatically the COBOL source code generated by NATSTAR on GCOS 7.

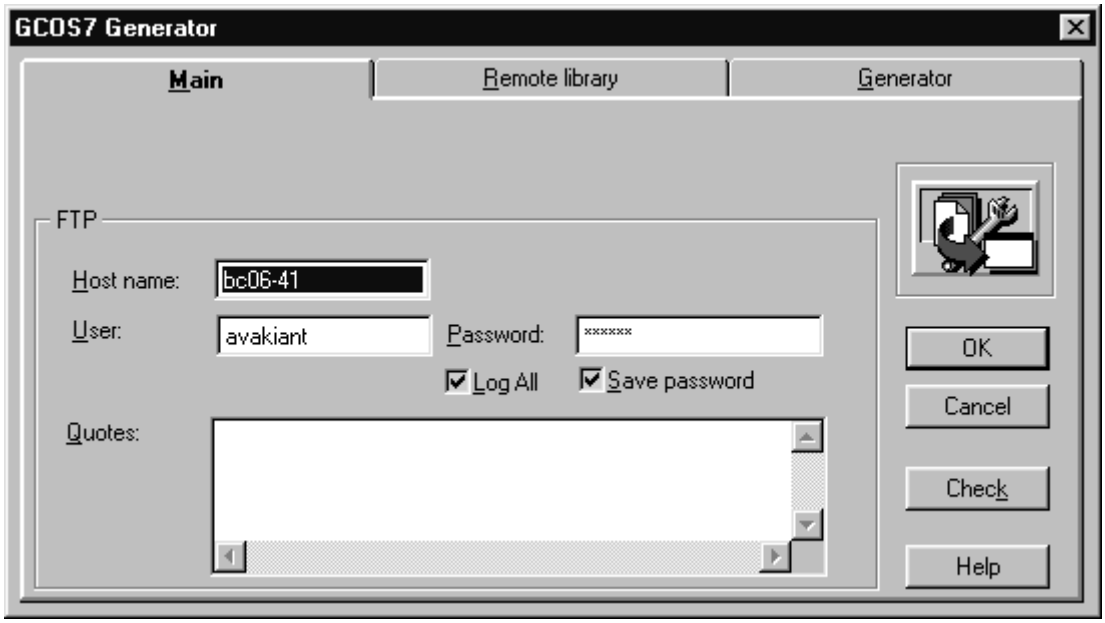

Figure 5-13. Main Tab

In the **Main** Tab, you specify:

Host name:	This is the name of the OPEN 7 on GCOS 7 on which the transfer is made. Note that this OPEN 7 must run on GCOS 7 on which the destination libraries are located.
User:	This is the user name on the OPEN 7.
Password:	This is the password of the user on the OPEN 7.
Save password:	This must be set if you want that the password are kept/save between two NATSTAR sessions (this parameter is introduced for confidentiality reasons only).
Log All:	This is useful if you want to see all messages about the transfer (recommended).

Quotes: This is the list of ftp commands to be performed after the connection with the OPEN 7.

CHECK button: This allows you to check the connection between NATSTAR and the ftp deamon on OPEN 7. You can press this button before the build of the application, to check the availability of the network.

Remote Library Tab

The **Remote Library** Tab allows you to define the libraries name on GCOS 7:

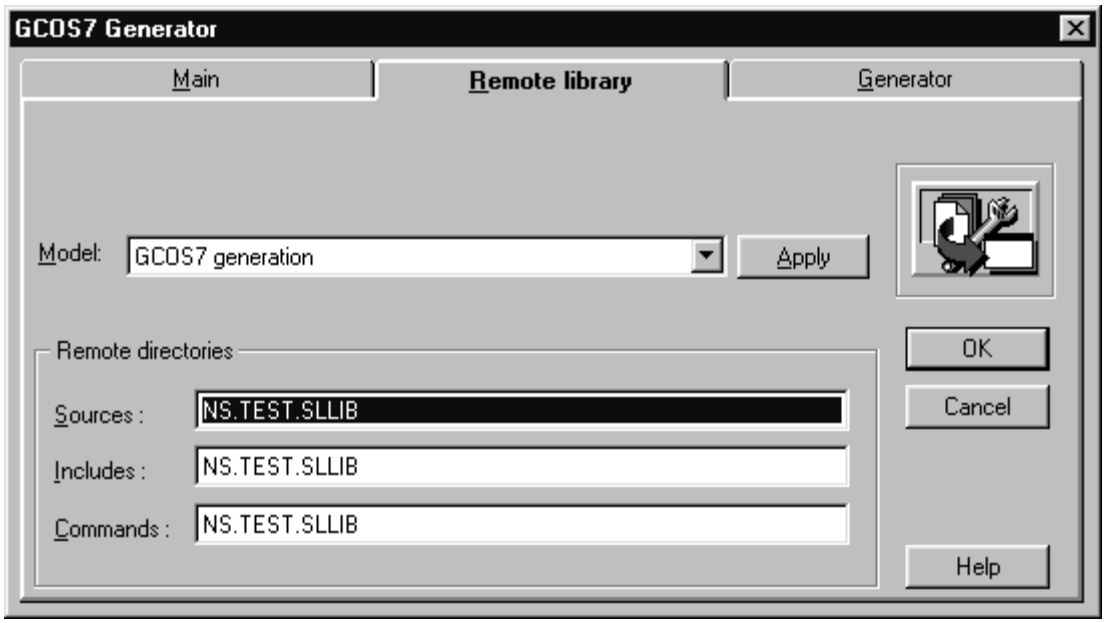

Figure 5-14. Remote Library Tab

In the **Remote Library** Tab, you specify:

Model: This must be set to "GCOS7 generation" for GCOS7/TDS. The combo box shows the different entries of DESCR in the section [GCOS7/TDS] in the file:

```
%NATSTAR%\GLOB\INI\ADE_RMGN.INI
```

These entries correspond to the type of generation.

For the moment, there is only one type of generation for [GCOS7/TDS].

APPLY button: This button allows you to validate the change about the Model.

Remote Directories:

You **must check** that the project of the user who runs OPEN 7 on GCOS 7 has the "**write**" **access right** on the libraries defined below. Otherwise, you must perform the command MDACL to give the "write" access right.

SOURCES: Enter the name of the source library which will contain the COBOL source generated by NATSTAR.

INCLUDES: Enter the name of the source library which will contain the COBOL copies generated by NATSTAR. This must be the same library as SOURCES, because it is input to the compilation (see the NSCOB7 JCL on GCOS 7).

COMMANDS: (reserved for further use)

Generator Tab

The **Generator** Tab allows you to define some characteristics of the generated COBOL.

Figure 5-15. Generator Tab

The default values are already proposed at the first time and you can keep these values which are adapted to COBOL 85 of GCOS 7.

MAX LINE SIZE: This is the number of characters containing in a COBOL line (default 80).

MAX IDENTIFIER LENGTH: This is the maximum number of characters for a COBOL identifier (default 31).

TABULATION: This is the step of tabulation (default 4).

GENERATOR OPTIONS: You can optionally indicate /CBLUNSIGNED to specify that the NUM(x,y) must be generate the COBOL type PACKED DECIMAL WITHOUT SIGN instead of the default generation which is PACKED DECIMAL SIGNED.

Server Configuration Directories

You can specify the directories located on your PC for the generation.

However, you are **recommended not to use** this functionality.

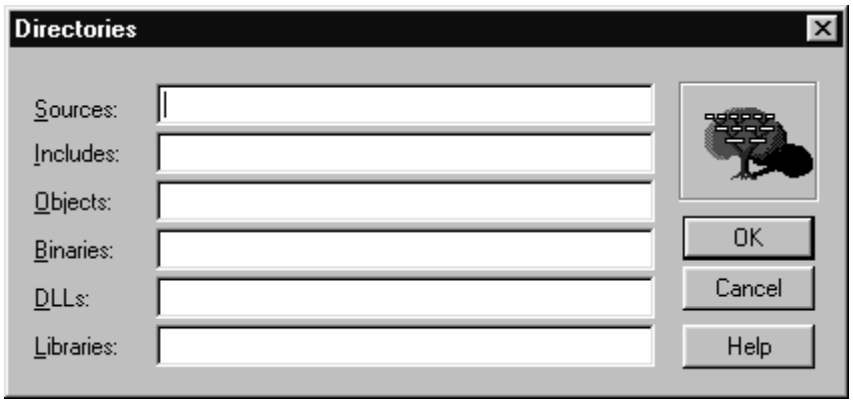

Figure 5-16. Server Configuration Directories

Modifying Library Description

Select a library, then do EDIT/MODIFY:

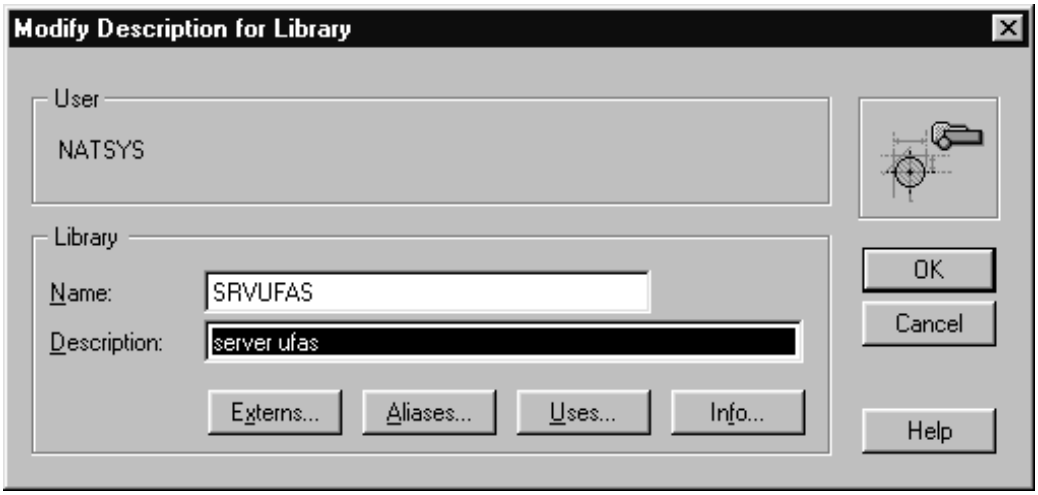

Figure 5-17. Modify Library Description

The following window allows you to define the EXTERNS... : (push the EXTERNS button).

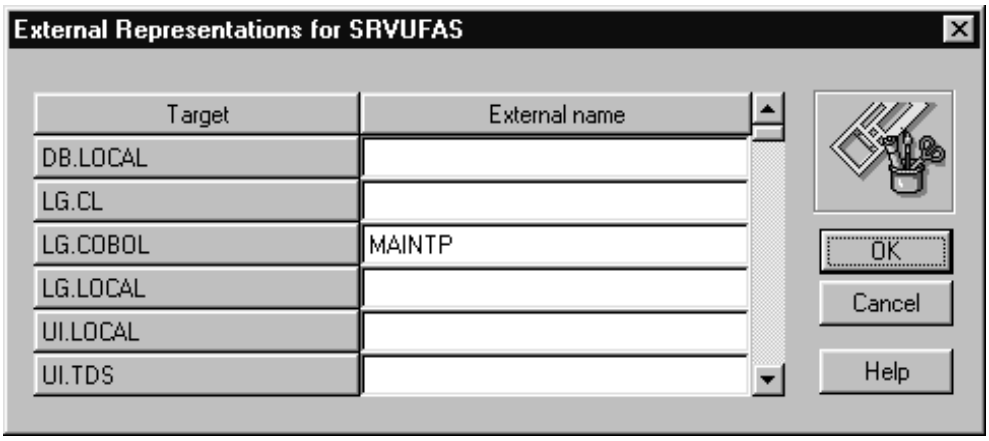

Figure 5-18. Modify External Representations

The external name for:

DB.LOCAL:	unused
LG.COBOL:	This is the name of the main entry point of the COBOL generated. It is also the name of the first TPR of the transaction for TDS. It is limited to 6 characters. **Note:** You can add a prefix by editing the files `%NATSTAR%\ini\gcos7.mkt` and `%NATSTAR%\Glob\ini\gcos7.mkt` in the COBOL statement: PROGRAM-ID.
LG.LOCAL:	unused
UI.LOCAL:	unused
UI.TDS:	unused

Make/Build Server Resources

When you build the server libraries SPECIFY "**Generate DLL**" to generate the COBOL source from the NCL instructions and/or functions, and also transfer all sources to GCOS 7 via OPEN 7.

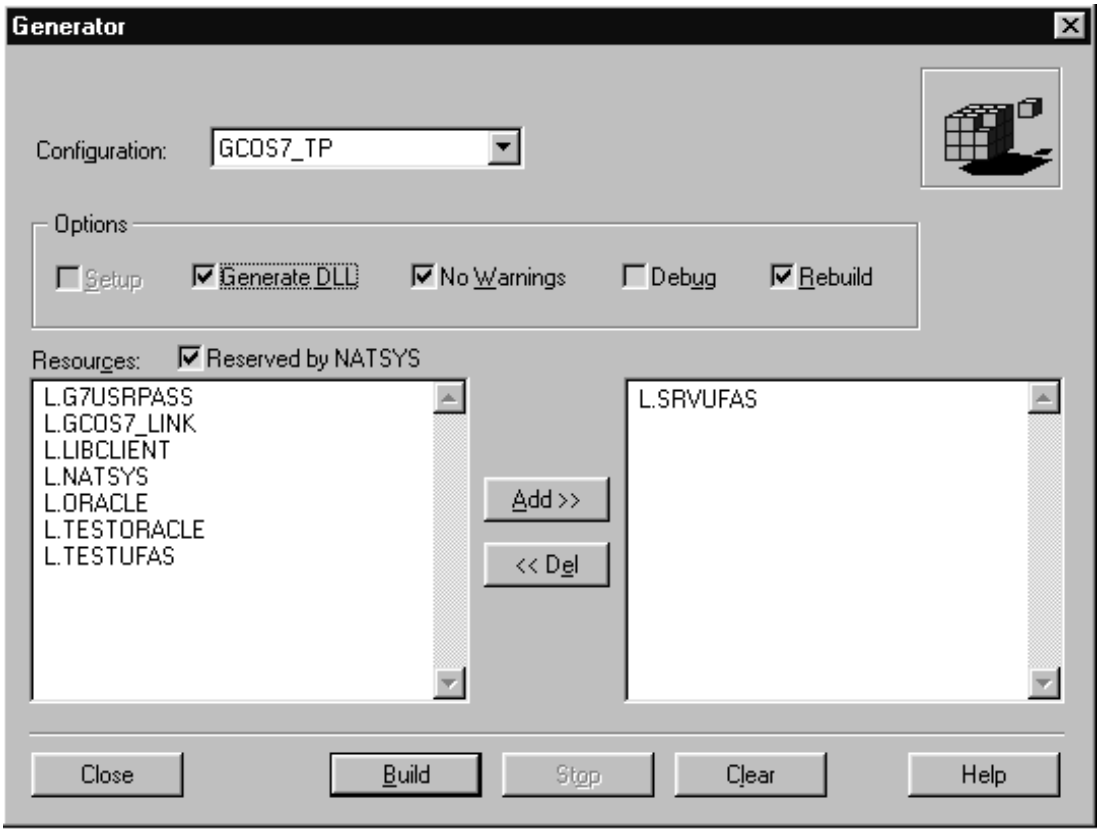

Figure 5-19. Modify External Representations

Generating Additional NCL

You define client/server logic by writing code in NCL. However, a complete client/server application requires additional NCL to control the Remote Procedure Calls that link the client and server. The RPC-related NCL, and other NCL for "housekeeping" purposes, is generated automatically by NatStar. This NCL must be generated, as described below, before you can generate COBOL. The generated NCL is added to the NCL you coded to create a complete set of NCL, from which you can generate COBOL.

Before you generate this additional NCL, you need to make choices that determine whether the generated code will become a NatStar TPR or not.

Defining an Instruction or Function To Be Local or Remote

Each instruction or function is defined as **Local** or **Remote**. Only GCOS 7 server procedures (instruction or function) **that will be called by a client via RPC** are defined as Remote. All other GCOS 7 server procedures, and all client procedures, are defined as Local. The default is Local, so you need to explicitly define only Remote procedures.

The following figure illustrates how procedures are defined as Local or Remote on the NatStar development workstation.

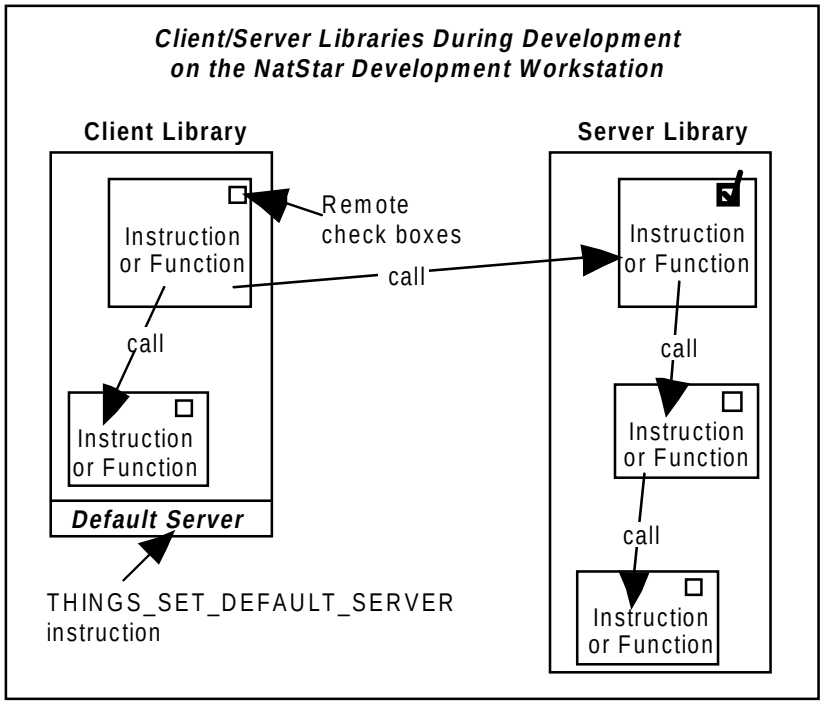

Figure 5-20. Defining a Function or Instruction as Local or Remote

In Figure 5-20, only one server procedure on GCOS 7 is called by the client, and so that procedure is defined as Remote. As shown in the figure, the "Remote" check box for that server procedure must be checked.

The figure includes two Local client procedures and two Local server procedures. Because the Remote boxes are not checked, by default these procedures are defined to be Local.

GCOS 7 code created in NatStar to be used for any purpose other than as a NatStar TPR -- for example, as a batch program -- is also defined as Local, because this code will never be called by a NatStar client.

The status of the Remote check box is also related to the use of the NCL instruction `THINGS_SET_DEFAULT_SERVER`. This instruction, which is described earlier in this chapter and illustrated in the appendices' example, determines whether the client will call server code locally (on the same system) or remotely. If you include the `THINGS_SET_DEFAULT_SERVER` instruction in the client *and* check the Remote box in the main procedure of the server TPR code, then the client initiates a Remote Procedure Call (RPC) to the remote server. You can check the Remote box but omit the `THINGS_SET_DEFAULT_SERVER` instruction from the client, and in this case the client will call the server locally. This feature can be useful during testing of client/server applications.

Using the Remote Check Box in Code Generation

Before beginning any code generation, make the following selections.

In the **Modify function or instruction window** for each function or instruction that you want to code generate, check the **Remote** box if you want to generate a remote procedure for use as a NatStar TPR (see Figure 5-21, which is from the appendix example application). Checking the Remote box causes the NatStar code generator to generate additional code that is used to handle RPC interactions and perform additional "housekeeping" functions.

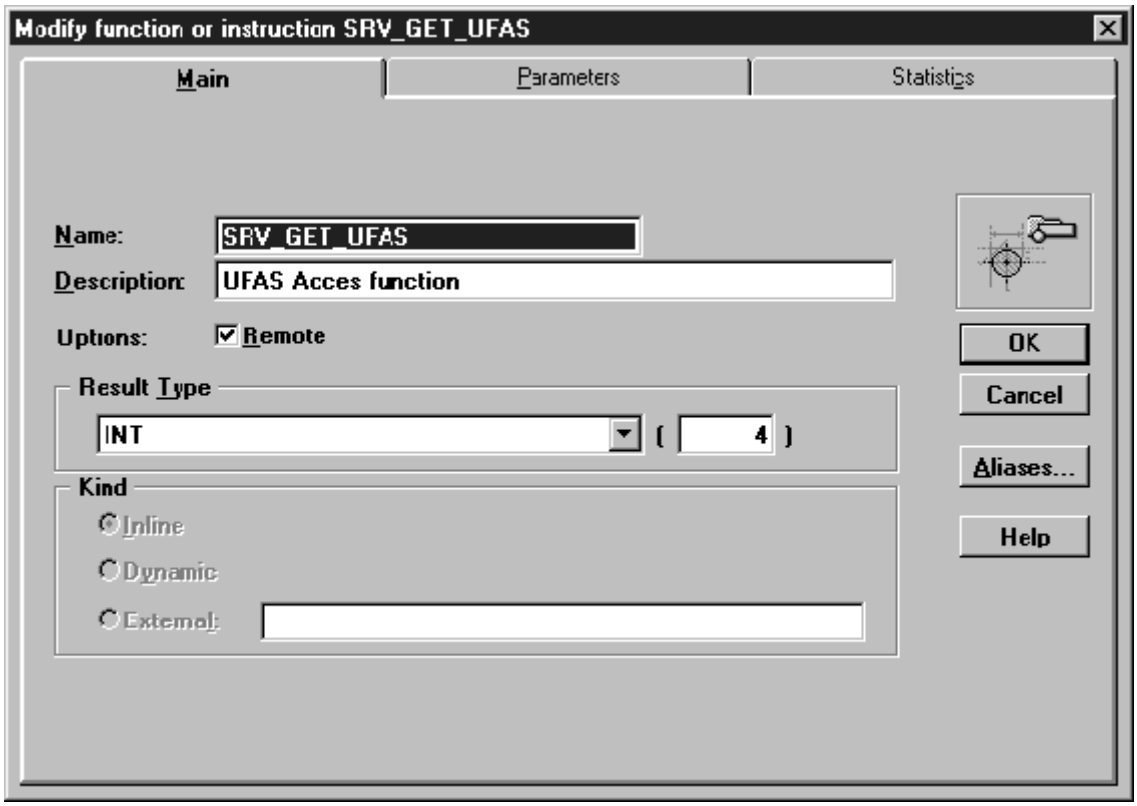

Figure 5-21. Check Remote to Generate Code for a NatStar TPR

To generate GCOS 7 source for a Local procedure, be sure that the **Remote** box is unchecked.

NCL Generation and COBOL Generation

The COBOL generator in the current release is fully integrated with NatStar. You must generate the NCL, then, the COBOL generator is automatically called.

5.4.2 Generating COBOL Source Code

You need to run the NCL generator to create RPC code and complete other internal NatStar processing.

To generate the NCL and the related COBOL source, select **Build resources...** from the **Make** menu, or click on the **Build** icon. When the **Generator** dialog box appears:

1. Select the server library or libraries and move them to the right side by clicking on the **Add>>** push button (see Figure 5-22, which uses the library names from the appendix example).
2. Be sure that the **Debug** box is ***not*** checked before NCL generation. Usually the **Debug** box is dimmed and cannot be checked, but be sure to verify its status. Check that the **Generate DLL** box **is** checked to indicate that the COBOL generation must be called.
3. Select the related GCOS 7 Configuration (as described earlier in Section 5.4.1 *Background Information*), so that the resulting source will be generated for GCOS 7. In Figure 5-22, the configuration is named **SRVUFAS**.
4. To generate NCL, then the specific COBOL for GCOS 7, and to transfer all the COBOL sources to the specified GCOS 7 platform, click on the **Build** button.

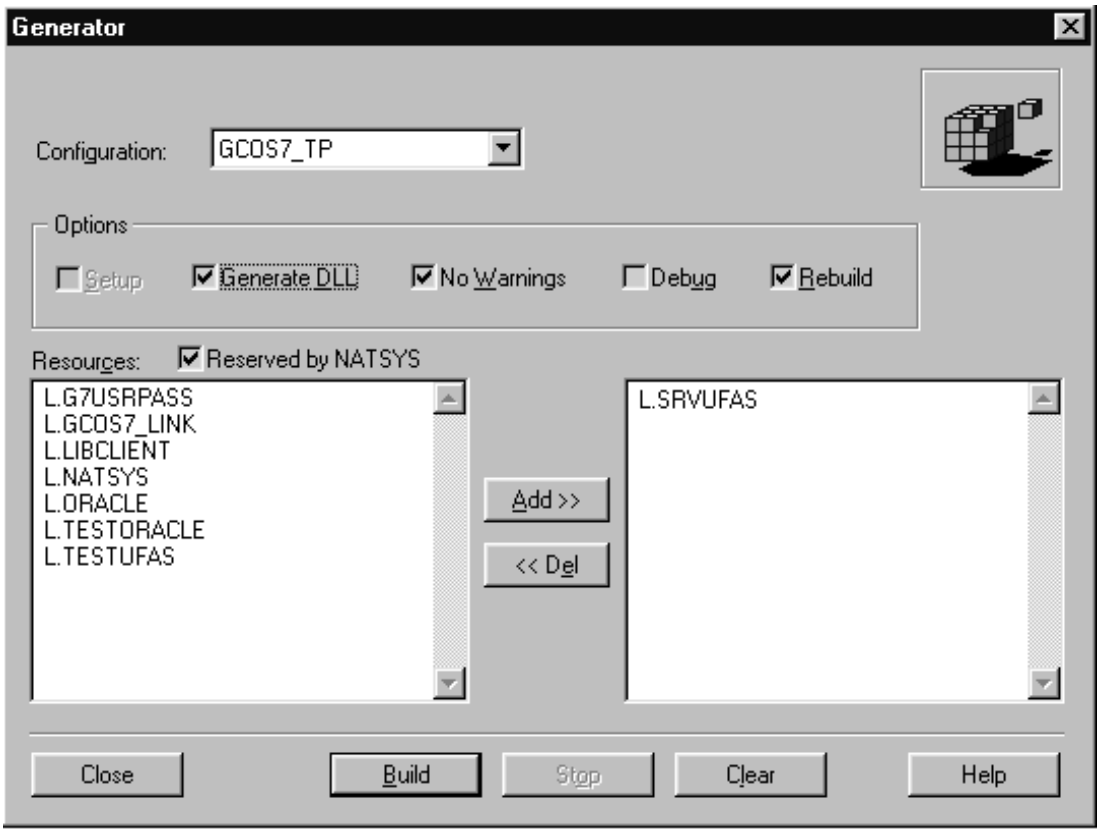

Figure 5-22. Generating NCL and Specific COBOL for GCOS 7

When the generation is complete, the text in the Status line at the bottom of the NatStar window stops changing. Then close the **Generator** dialog box by clicking on **Close**.

To check for errors, select the **Log** entry from the **Windows** menu, or double-click on the **Log** icon to view the Log file. If no errors are listed in the Log window, exit from NatStar.

IMPORTANT:

If you use a COBOL reserved word when coding in NCL in a program that will be code generated in COBOL, an error will occur when the reserved word is encountered during processing of the source code by the GCOS 7 COBOL-85 compiler. To avoid this problem, avoid the use of COBOL reserved words as the names of NCL instructions, functions, and parameters, and in defining variables; i.e., variables to be listed in the LOCAL clause.

5.4.2.1 COBOL Generator Output

When the COBOL Generator has executed without fatal errors, the generated COBOL-85 source can be found in the GCOS 7 libraries specified in the GCOS 7 Configuration defined for the Build.

The COBOL generator creates the following files:

`N_xxxxxx_CBL`	COBOL source file produced by the generator. This includes the translated NCL code, any native code included with the NCL, and the additional source code generated automatically by NatStar.
`NZxxxxxx-CPY`	COBOL source COPY file required by the COBOL program. This COPY file (which is not shown in the COBOL Generator window) is referenced in the `N_xxxxxx_CBL` source file by the statement: `COPY NZxxxxxx-CPY`

The `xxxxxx` in the file names is derived from the name of the NatStar library for which code was generated.

5.4.2.2 COPY Files Provided

NOTE:
The information in this section is provided to assist you, if you are interested in studying the generated COBOL code. This information is not required during the normal development of a Bull/NatStar application.

Data types (integers, float, etc.) are defined (perhaps differently) for each COBOL variant supported by the NatStar COBOL Generator (e.g., GCOS 7 COBOL-85, Micro Focus COBOL). At compile time, the appropriate definitions (e.g., for GCOS 7 COBOL-85) are provided by a generated COPY statement that implements a COBOL-85 REPLACE statement. This REPLACE statement translates all non-portable COBOL constructions into the one used by the target COBOL compiler (e.g., COBOL-85 compiler on GCOS 7).

To implement this REPLACE statement, all COBOL modules generated by NatStar begin with the COBOL header shown below:

```
****************************************************************
* module N                                                     *
* Program generated with Nat Systems Code Generator            *
* 1.00.000                                                     *
* Copyright (C) 1996 Nat Systems. All Rights Reserved.         *
*                                                              *
****************************************************************

IDENTIFICATION DIVISION.
  COPY NSCOBOL-CPY IN INLIB1.
  COPY TERMREPL-CPY IN INLIB1.
```

NOTE:
The COPY file for TERMREPL and NSCOBOL is ***not*** generated by the NatStar COBOL generator. This COPY file is delivered with the NATSTAR7 product and resides on GCOS 7.

5.5 Compiling a NatStar TPR

This section describes how to compile the COBOL-85 source code generated by NatStar for GCOS 7.

5.5.1 Compiling NatStar TPR Code

You do not need to modify any of the files generated by the NatStar COBOL code generators before using these files as input to the compilers on GCOS 7. All of these files can be used without modification.

NOTE:
Never modify any of the generated code. Instead, always make changes to the NCL and re-generate the COBOL.

5.5.2 Compiling COBOL-85 Code

There are several ways to compile the COBOL code generated by NatStar (refer to *COBOL-85 User's Guide*, and the *TDS System Programmer's Guide*). An example is provided at the end of this section.

To compile the NatStar-generated COBOL source, the COBOL-85 compiler requires:

- **NatStar-generated COBOL program file and NatStar-generated COBOL COPY file.** These are the files described earlier in Section 5.4.2.1 *COBOL Generator Output* that have been automatically transferred to GCOS 7. You must place the generated COBOL program file and the COPY file in the same GCOS 7 libraries.
- **Output object module file.** This file is written by the COBOL-85 compiler. After a successful compilation, this file will contain an object module in a format that is acceptable to the LINKER step described in Section 5.6 *Linking a NatStar TPR* later in this chapter.

The JCL for COBOL compilation is NSCOB7, and it is located in SI7.NS7.SLLIB. Read the comments on this JCL in the subfile.

The following listing shows the NSCOB7 parameters.

```
COMM '*******************************************************************************';
COMM '*    NSCOB7 V6           SUBFILE CONTAINS SOURCE JCL TO :                      *';
COMM '*               Compile COBOL of the NATSTAR generated COBOL source.          *';
COMM '*                                                                             *';
COMM '*                                                                             *';
COMM '*    PARAMETERS :                                                             *';
COMM '*    ----------                                                               *';
COMM '*                                                                             *';
COMM '*      Positional :                                                           *';
COMM '*      ----------                                                             *';
COMM '*                                                                             *';
COMM '*       1) ***MANDATORY PARAMETER***                                          *';
COMM '*          Source name of the COBOL source generated by NATSTAR, this name UST NOT   *';
COMM '*          contains the characters "." and must be replaced by "_" for example.      *';
COMM '*                                                                             *';
COMM '*       2) Optional PARAMETER                                                 *';
COMM '*          Cobol compilation option (for DEBUG) for example. Default value :Empty.   *';
COMM '*                                                                             *';
COMM '*                                                                             *';
COMM '*       3) Optional PARAMETER                                                 *';
COMM '*          Must be "FRM" to ask the Generation of the COBOL DPS7 from COBOL source   *';
COMM '*          generated by NATSTAR : it is the Default value.                    *';
COMM '*          Can be "NFRM" to skip this Generation with NATSTAR version 2.15    *';
COMM '*          Default value : "NFRM"                                             *';
COMM '*                                                                             *';
COMM '*      Keywords :                                                             *';
COMM '*      --------                                                               *';
COMM '*                                                                             *';
COMM '*       MODE) ***MANDATORY PARAMETER*** if {Optionnal PARAMETER 3} = FRM      *';
COMM '*             Must be "TDS" or "BATCH".                                       *';
COMM '*             This parameter indicates the dimension in which the program will be   *';
COMM '*             executed.                                                       *';
COMM '*             Default value : "TDS"                                           *';
COMM '*                                                                             *';
COMM '*       MAIN) ***MANDATORY PARAMETER***  If MODE=BATCH                        *';
COMM '*             It is the name of the BATCH program to be compiled.             *';
COMM '*                                                                             *';
COMM '*       INLIB) ***MANDATORY PARAMETER***                                      *';
COMM '*              Library Name that contains the Source name of the COBOL source       *';
COMM '*              generated by NATSTAR.                                          *';
COMM '*              This Library must also contains the COPY file generated by NATSTAR,   *';
COMM '*              these copy files must be renamed <copy-name>-CPY instead of    *';
COMM '*               <copy- me>.cpy                                                *';
COMM '*                                                                             *';
COMM '*                                                                             *';
COMM '*       INLIBDVC) Optional PARAMETER                                          *';
COMM '*                 Device Class of the INLIB.                                  *';
```

```
COMM '*                 Default value : Empty.                                        *';
COMM '*                                                                               *';
COMM '*       INLIBMD) Optional PARAMETER                                             *';
COMM '*                Media Name of the INLIB.                                       *';
COMM '*                Default value : Empty.                                         *';
COMM '*                                                                               *';
COMM '*       PRTLIB)  ***MANDATORY PARAMETER***                                      *';
COMM '*                Library Name that contains the listing result for the          *';
COMM '*                generation and the COBOL compilation.                          *';
COMM '*                                                                               *';
COMM '*       PRTLIBDV) Optional PARAMETER                                            *';
COMM '*                 Device Class of the PRTLIB.                                   *';
COMM '*                 Default value : Empty.                                        *';
COMM '*                                                                               *';
COMM '*       PRTLIBMD)  Optional PARAMETER                                           *';
COMM '*                 Media Name of the PRTLIB.                                     *';
COMM '*                 Default value : Empty.                                        *';
COMM '*                                                                               *';
COMM '*       CULIB)     ***MANDATORY PARAMETER***                                    *';
COMM '*                 Library Name that contains the Compiles Unit resulting        *';
COMM '*                 from the COBOL compilation.                                   *';
COMM '*                                                                               *';
COMM '*       CULIBDVC) Optional PARAMETER                                            *';
COMM '*                Device Class of the CULIB                                      *';
COMM '*                Default value : Empty.                                         *';
COMM '*                                                                               *';
COMM '*       CULIBMD)  Optional PARAMETER                                            *';
COMM '*                Media Name of the CULIB.                                       *';
COMM '*                Default value : Empty.                                         *';
COMM '*                                                                               *';
COMM '*       NSSLLIB)  Optional PARAMETER                                            *';
COMM '*                Library Name that contains the delivered source file for NATSTAR.  *';
COMM '*                Default value : SI7.NS7.SLLIB.                                 *';
COMM '*                                                                               *';
COMM '*       NSSLLIBD)   Optional PARAMETER                                          *';
COMM '*                  Device Class of the NSSLLIB                                  *';
COMM '*                  Default value : Empty.                                       *';
COMM '*                                                                               *';
COMM '*       NSSLLIBM)  Optional PARAMETER                                           *';
COMM '*                 Media Name of the NSSLLIB.                                    *';
COMM '*                 Default value : Empty.                                        *';
COMM '*                                                                               *';
COMM '*       TMPLIB1)   Optional PARAMETER                                           *';
COMM '*                 Library Name for the Edition phase generation                 *';
COMM '*                 Default value : "TEMP"                                        *';
COMM '*                                                                               *';
COMM '*********************************************************************************';
```

EXAMPLE:

The following sample JCL illustrates how to compile COBOL-85 source code into an object module, which you can link to create a NatStar TPR.

```
EJR NSCOB7 LIB=SI7.NS7.SLLIB
VL = (NSRVU9N_CBL, INLIB=NS.TEST.SLLIB,
PRTLIB=NS.TEST.LISLIB, CULIB=NS.TEST.CULIB) HOLDOUT;
```

❑

5.6 Linking a NatStar TPR

When you have completed an error-free compilation of the code generated by NatStar, you are ready to link your generated object module with a set of system libraries. The result will be a SM Transaction that can be used as a NatStar TPR in TDS.

The link can be made using the job NSLK7 delivered with ISI7 in the NatStar package in the SI7.NS7.SLLIB library. Please refer to the comments about the JCL in this subfile.

The following listing shows the NSLK7 parameters:

```
COMM '*******************************************************************************';
COMM '*    NSLK7 V2 (Jan 30, 1998) SUBFILE CONTAINS SOURCE JCL TO :                 *';
COMM '*       1) If necessary, prepare the LINKER COMMAND from the LINKER COMMAND   *';
COMM '*          produced by TDS.                                                   *';
COMM '*       2) Link the TPR or the MAIN PROGRAM generated with the NATSTAR COBOL  *';
COMM '*          generator.                                                         *';
COMM '*                                                                             *';
COMM '*                                                                             *';
COMM '*    PARAMETERS :                                                             *';
COMM '*    ----------                                                               *';
COMM '*                                                                             *';
COMM '*      Positional :                                                           *';
COMM '*      ----------                                                             *';
COMM '*                                                                             *';
COMM '*       1) ***MANDATORY PARAMETER***                                          *';
COMM '*          Name of the TPR which is the first TPR of the NATSTAR Transaction, OR *';
COMM '*          Name of the MAIN PROGRAM for a BATCH Load Module.                  *';
COMM '*                                                                             *';
COMM '*      Keywords :                                                             *';
COMM '*      --------                                                               *';
COMM '*                                                                             *';
COMM '*       LINK) Optional Parameter                                              *';
COMM '*             IF TYPE=SM : Name of the LINKER COMMAND produced by TDS.        *';
COMM '*             IF TYPE=LM : Name of the LINKER COMMAND to produced a Batch program. *';
COMM '*                          It is BATCH_LNK  for a sample Batch.               *';
COMM '*                          It is ORACLE_LNK for a Batch which used ORACLE.    *';
COMM '*             Default value : "TP7LINKTPR"                                    *';
COMM '*                                                                             *';
COMM '*       TYPE) Optional Parameter                                              *';
COMM '*             Must be "SM" or "LM".                                           *';
COMM '*             In TDS environment   : to produce a TPR you must specified "SM" *';
COMM '*             In BATCH environment : to produce a Load Module, you must specify "LM" *';
COMM '*             Default value : "SM"                                            *';
COMM '*                                                                             *';
COMM '*        USE) Optional Parameter                                              *';
COMM '*             Must be "NONE" or "ORACLE"                                      *';
```

```
COMM '*             When the LM or SM uses the ORACLE interface you must specify          *';
COMM '*             USE=ORACLE                                                            *';
COMM '*             This parameter allows to reduce the LM or SM produced when ORACLE is  *';
COMM '*             not used.                                                             *';
COMM '*             Default value : "NONE"                                                *';
COMM '*                                                                                   *';
COMM '*                                                                                   *';
COMM '*       LNKSL)  ***MANDATORY PARAMETER***                                           *';
COMM '*               IF TYPE=SM : Library name which contains the LINKER COMMAND produced *';
COMM '*               by TDS.  It is the <TDS-NAME>.SLLIB                                 *';
COMM '*               IF TYPE=LM : Library name which contains the LINKER COMMAND to      *';
COMM '*                            produce a Batch program.                               *';
COMM '*                            It is the SI7.NS7.SLLIB                                *';
COMM '*               Default value : Empty.                                              *';
COMM '*                                                                                   *';
COMM '*       LNKSLDVC)    Optional Parameter                                             *';
COMM '*                    Device class of the LNKSLLIB                                   *';
COMM '*                    Default value : Empty.                                         *';
COMM '*                                                                                   *';
COMM '*       LNKSLMD)     Optional Parameter                                             *';
COMM '*                    Media name of the LNKSLLIB                                     *';
COMM '*                    Default value : Empty                                          *';
COMM '*                                                                                   *';
COMM '*       OUTLIB) ***MANDATORY PARAMETER***                                           *';
COMM '*               Sharable Module Library name which contains the TPR's SM.           *';
COMM '*               It is the <TDS-NAME>.SMLIB                                          *';
COMM '*                                                                                   *';
COMM '*       OUTLIBDV)  Optional Parameter                                               *';
COMM '*                  Device class of the OUTLIB                                       *';
COMM '*                  Default value : Empty.                                           *';
COMM '*                                                                                   *';
COMM '*       OUTLIBMD) Optional Parameter                                                *';
COMM '*                 Media name of the OUTLIB                                          *';
COMM '*                 Default value : Empty                                             *';
COMM '*                                                                                   *';
COMM '*       PRTLIB) ***MANDATORY PARAMETER***                                           *';
COMM '*               Library name which contains the result listing of the linker.       *';
COMM '*               It is the <TDS-NAME>.EDITION                                        *';
COMM '*                                                                                   *';
COMM '*       PRTLIBDV)  Optional Parameter                                               *';
COMM '*                  Device class of the PRTLIB                                       *';
COMM '*                  Default value : Empty                                            *';
COMM '*                                                                                   *';
COMM '*       PRTLIBMD) Optional Parameter                                                *';
COMM '*                 Media name of the PRTLIB                                          *';
COMM '*                 Default value : Empty                                             *';
COMM '*                                                                                   *';
COMM '*       CULIB)     ***MANDATORY PARAMETER***                                        *';
COMM '*                  Library Name that contains the Compile Unit resulting from the   *';
COMM '*                  COBOL compilation.                                               *';
COMM '*                                                                                   *';
```

```
COMM '*       CULIBDVC) Optional Parameter                                                *';
COMM '*                 Device Class of the CULIB                                         *';
COMM '*                 Default value : Empty.                                            *';
COMM '*                                                                                   *';
COMM '*       CULIBMD)  Optional Parameter                                                *';
COMM '*                 Media Name of the CULIB.                                          *';
COMM '*                 Default value : Empty.                                            *';
COMM '*                                                                                   *';
COMM '*       NSCULIB)  Optional Parameter                                                *';
COMM '*                 Library Name that contains the delivered compile units for NATSTAR.*';
COMM '*                 Default value : SI7.NS7.CULIB                                     *';
COMM '*                                                                                   *';
COMM '*       NSCULIBD)  Optional Parameter                                               *';
COMM '*                   Device Class of the NSCULIB                                     *';
COMM '*                   Default value : Empty.                                          *';
COMM '*                                                                                   *';
COMM '*       NSCULIBM)  Optional Parameter                                               *';
COMM '*                  Media Name of the NSCULIB.                                       *';
COMM '*                  Default value : Empty.                                           *';
COMM '*                                                                                   *';
COMM '*************************************************************************************';
```

Example of usage of NSLK7.

EXAMPLE:

```
ejr nslk7 lib=si7.ns7.sllib
  vl=(mainufastpr,lnksl=scha.sllib,outlib=scha.smlib,
    prtlib=scha.edition,culib=ns.test.culib) holdout
```

This job is located in the SI7.NS7.SLLIB file delivered with the NatStar product in this directory and requires the SI7.NS7.CULIB file to resolve linking references.

❑

5.7 Testing Bull/NatStar Applications

Testing NatStar applications requires careful planning, because you will need to test logic that will execute on the client workstations with NatStar-generated logic and conventional logic that will execute on GCOS 7 under TDS and access UFAS files and/or other GCOS 7 data. This section discusses the methods that can be used to test and debug NatStar applications.

5.7.1 Debugging on the NatStar Development Workstation

You can do initial debugging in the NatStar development environment. This environment allows you to execute NCL code interpretively, and optionally to execute the code in step (debug) mode. This type of testing can identify logic errors in the NCL source. However, only NCL code is tested, so native code and calls to GCOS 7 conventional subroutines cannot be tested.

NOTE:
The problems associated with testing native code embedded in NCL programs is another reason to avoid the use of native code if practical.

You can continue debugging the client NCL interpretively even after you have generated the GCOS 7 code, exported it to GCOS 7, then compiled and linked it as a NatStar TPR. You do not need to generate code for the client application until you are ready to distribute copies of the code to multiple client systems for production operation.

5.7.2 Simulating GCOS 7 Native and Conventional Code

If the native code and called subroutines that will execute on GCOS 7 are not too lengthy and/or complex, you can simulate this logic in NCL and use the simulated logic in testing on the development workstation. You can create a local library that contains the GCOS 7 logic, and generate the NCL as Local. By defining these instructions as Local, they will be executed when the client logic is executed during testing.

One way to simulate data-access instructions is to prompt for data input from the keyboard: you can then provide the data during debugging.

Another way to simulate data-access instructions is to use the NatStar relational database manager to access data stored locally. If you structure client/server applications so that database-access code is isolated and contained in a user library, you can use one version of that library to access a local database for test purposes and another version of that library to access GCOS 7 databases.

Testing using the NatStar development workstation is described in more detail in the *NatStar Development Guide -- Client-Server* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

5.7.3 Using GCOS 7 Debugging Tools

After compiling and linking the NatStar TPR on GCOS 7, as described earlier in this chapter, you can debug at least part of the logic of the server application using a source-language debugger such as PCF. Refer to the *PCF User's Guide* for information about how to use PCF.

To obtain traces of the NatStar session activity follow this command list:

1. Suppress all the files being in the TDSNAME.DEBUG library.
2. Connect to the TDS through a terminal with the same USERID, as in the NatStar application.
3. Launch the TRACE PRINT transaction.
4. Abnormally disconnect from the TDS with => $*$DIS.
5. Launch the application through the NatStar in the normal way.
6. Disconnect the TDS through NatStar.
7. Reconnect with a terminal under the same USERID.
8. Run the TRACE OFF transaction.
9. Print the subfile in the TDSNAME.DEBUG library.

You can put a session in TRACE mode by using the MASTER command:

```
send_tds 'TRACE PRINT TX=<NatStar TX> USER=*' TDS=<TdsName>;
```

and you can stop a session in TRACE mode by using the MASTER command:

```
send_tds 'TRACE OFF TX=* OPER=*' TDS=<TdsName>;
```

5.7.4 Debugging in a Simulated Production Environment

In the final stage of testing a Bull/NatStar application, the objective is to test the end-to-end client/server application, including the client interaction with the user and the server interaction with database(s) or file(s).

For this purpose, you will need to set up a TDS test and perhaps create a test database. These considerations are the same as when you are testing any TPR prior to beginning production use.

5.8 Generating, Compiling, and Linking Client CODE

When you have completed debugging the client/server application and are ready to distribute copies of the client code to multiple client system, you will need to:

- Generate NCL.
- Generate C source.
- Compile and link the C source.

These are standard operations that are described either in the Nat Systems documentation or in the documentation for your client C compiler and operating system.

6. NatStar in a GCOS 7 BATCH Environment

6.1 Preparing a Batch Program

The BATCH environment facility allows you to generate a program in a GCOS 7 batch dimension. It is produced by NatStar and is compiled, linked and executed on GCOS 7. Its execution is not activated by NatStar but by an operator and it runs without communication.

This chapter explains how to generate a COBOL program to be executed in a BATCH environment.

This is not the only possible method but it is the simplest.

It may be easiest to include or write all the instructions in a single NCL library, then to produce a single file containing all the instructions translated into COBOL.

Alternatively, you can write instructions which will be the main Batch program, and which will call other instructions.

1. Define these functions without the Remote option:

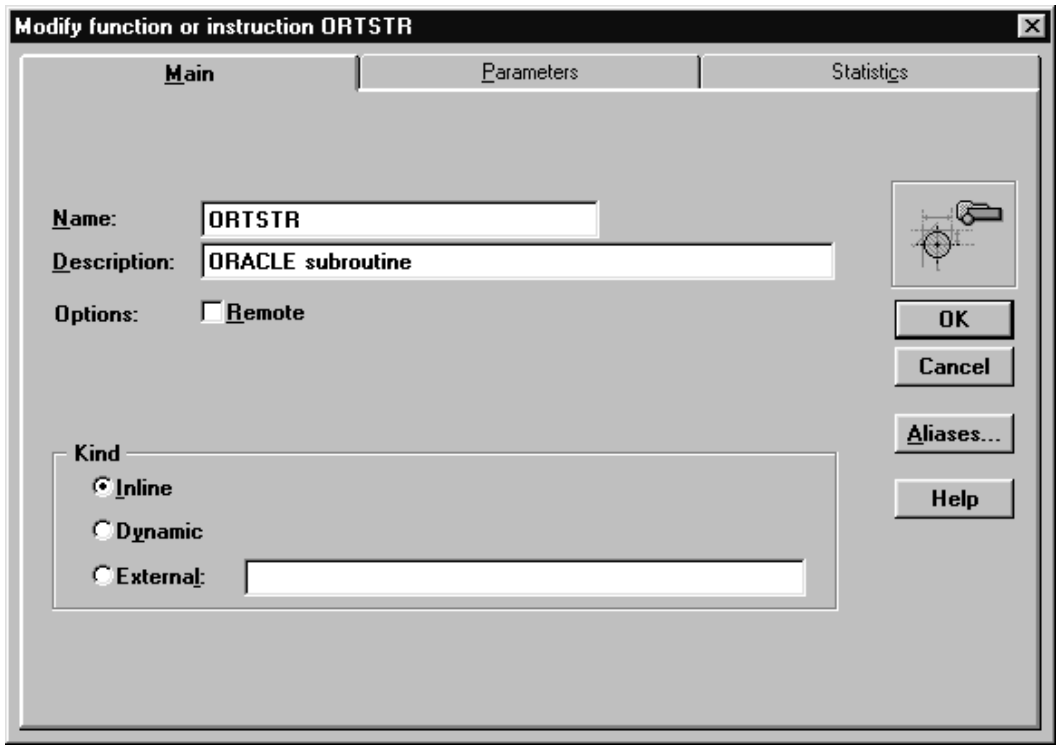

Figure 6-1. Modify Function

2. Define a SERVER configuration (Option menu / Configuration. Enter a configuration name, then select MISC...). The Main window should be:

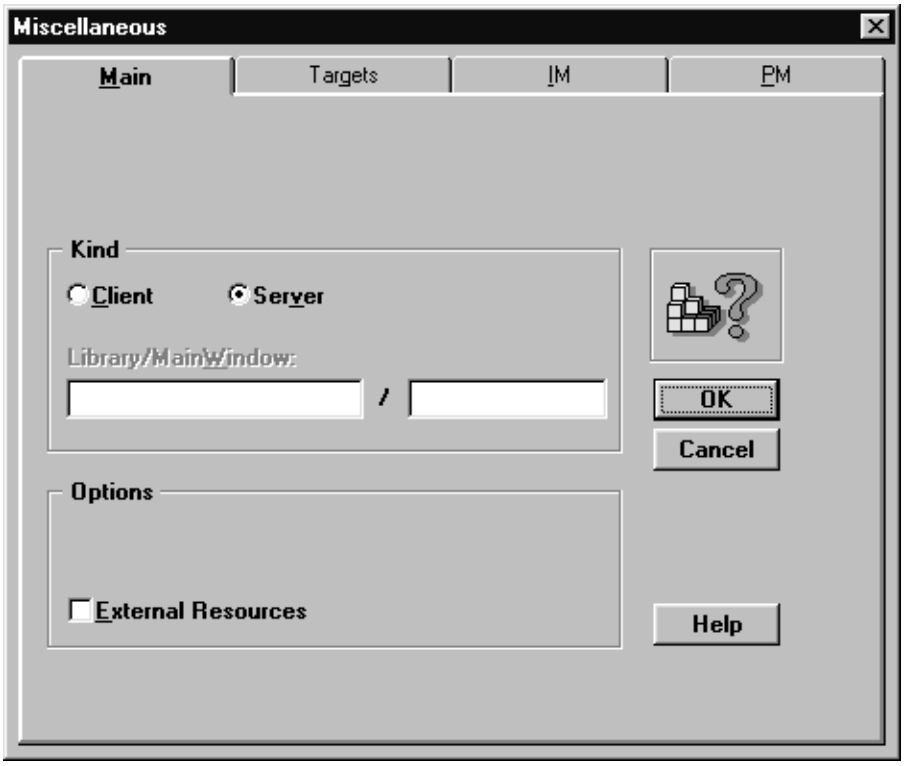

Figure 6-2. Main Window

3. In the Targets windows, set target UI to GCOS 7/TDS, set the others to LOCAL:

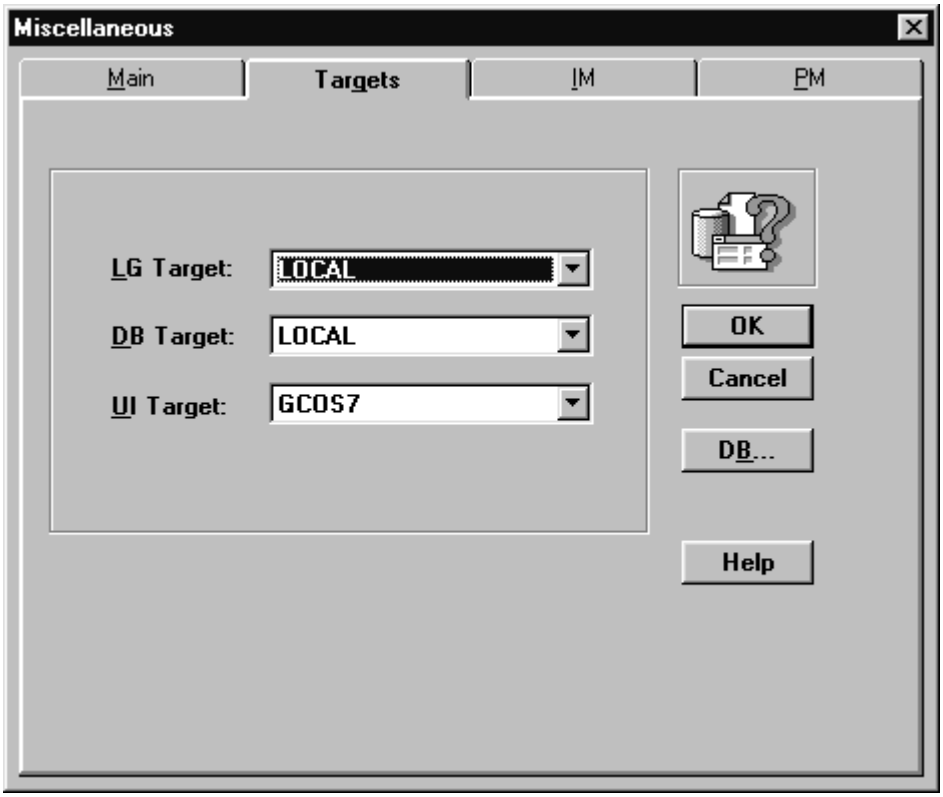

Figure 6-3. Targets Window

4. The IM window must not specify any transaction option:

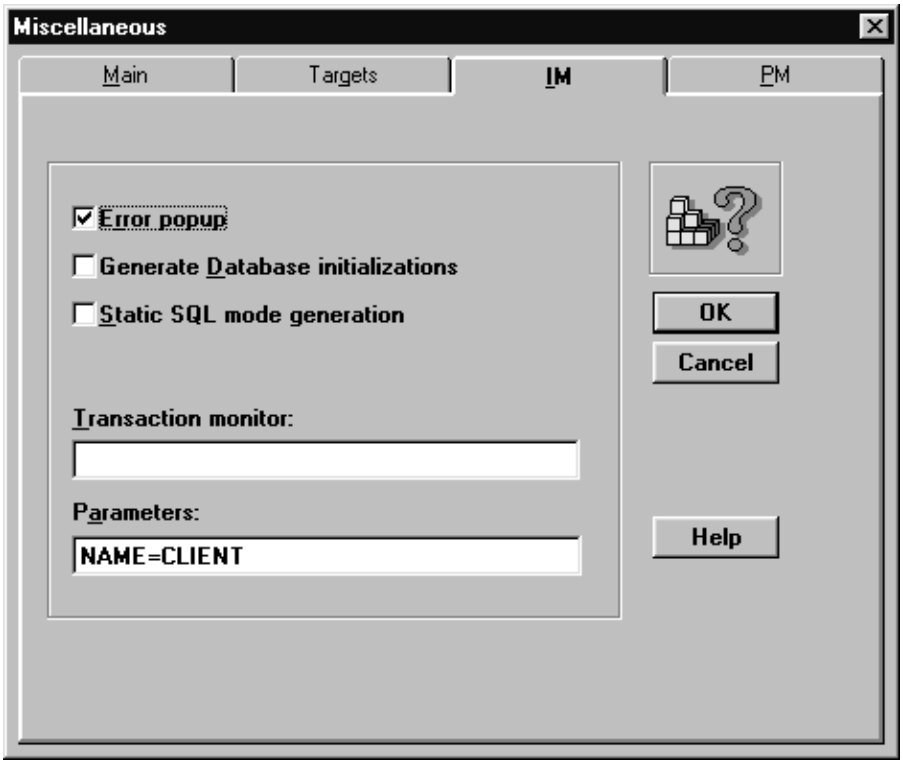

Figure 6-4. IM Window

The PM window is not significant.

5. BUILD the library with this server configuration.
6. The COBOL generator is activated automatically during Step 5.
7. The generated program is automatically transferred to GCOS 7 during Step 5.
8. Compile the COBOL source program on the DPS 7000:

   Use the JCL NSCOB7 to adapt the source code characteristics then compile it. This JCL is delivered in the library SI7.NS7.SLLIB.

   The significant parameters for a BATCH program are:

   – Positional 1: the name of the file containing the COBOL program,
   – MODE=BATCH: to adapt the generated program to a BATCH module,
   – MAIN=<*main entry point*>: to identify the main COBOL program.

The following parameters are mandatory:

- INLIB=<*source library*>: name of the library that contains the generated COBOL program.
- PRTLIB=<*listing library*>: name of the library that contains the listing result of the COBOL compilation.
- CULIB=<*compile unit library*>: name of the library that contains the Compile Units resulting from the compilation.

9. LINK the batch program using the JCL NSLK7 delivered in the library SI7.NS7.SLLIB.

   The significant parameters for BATCH program are:

   - Positional 1: name of the Main Program for a BATCH Load Module.
   - LINK=BATCH_LNK: for a BATCH program or LINK=ORACLE_LNK: if Oracle primitives are used.
   - TYPE=LM: to produce a load-module.

   The following parameters are mandatory:

   - LNKSL=SI7.NS7.SLLIB: must be specified for a BATCH Load Module.
   - OUTLIB=<*load module library*>: name of the library that contains the BATCH program resulting from the linker execution.
   - PRTLIB=<*listing library*>: name of the library that contains the listing result of the LINKER.
   - CULIB=<*compile unit library*>: name of the library that contains the Compile Units resulting from the COBOL compilation.

10. Execute the BATCH program.

6.2 Preparing an ORACLE Batch Program

The ORACLE facility allows you to generate a program in a GCOS 7 batch dimension that uses an ORACLE database. The program is produced by NatStar and it is compiled, linked and executed on GCOS 7.

The program is generated as a BATCH program with the following characteristic; the JCL NSLK7 delivered in the library SI7.NS7.SLLIB is called with the same parameters as a BATCH program, but with the additional parameter USE=ORACLE.

Before running the BATCH/ORACLE program, you must start the ORACLE server on GCOS 7 or another platform, but the client part of ORACLE V7 must be available in GCOS 7.

Note that a BATCH/ORACLE program is generated with an ORACLE OCI call.

For further information, refer to the NatStar documentation: *Programmatic Interface - Data Base Access*.

7. ORACLE Support in TDS/BATCH Mode

7.1 NatStar for GCOS7 with Oracle Support

Oracle support consists of:

- the NatSystems Oracle driver port on the GCOS 7 Oracle system,
- the implementation of the SQL_EXEC verbs in generated COBOL.

This OCI interface can be used on GCOS 7 in batch mode as well as under TDS.

OCI means that the SQL_EXEC NCL interface in NatStar is supported:

- Users can write NCL code using SQL_EXEC verbs.
- Those verbs will be converted into an OCI call on the GCOS 7 target.
- Code using SQL_EXEC verbs behaves the same in GCOS 7 Oracle as in an AIX Oracle instance.

7.2 No ORB Support

Oracle support does not mean ORB support. The NatStar Object engine does not exist on TDS or in Batch GCOS 7.

This means that:

- You cannot receive, create, send or apply methods to an object on the GCOS 7 machine.
- You cannot use instructions nor can you use function manipulating objects.

7.3 Access Using RPC

All access to the Oracle Database must be made using RPC mode.

NatStar classes cannot be generated on that target.

NatStar Remote Instruction (RPC on GCOS 7) uses a fixed length parameter list only.

You can still use NatStar IM when a GCOS 7 server is present in the architecture, but IM will not be available on the GCOS 7 server itself.

Two-Tier Architecture

In other words, in a 2 tier architecture, IM can be used on the client server and implement Oracle access by plugging remote instruction calls within the local object using NatStar CDAMs and then hide the database access from the application programs (CDAM is a way for overwriting NatStar IM Database access).

Three-Tier Architecture

Also, in a 3 tier architecture this can be done on the first tier but also on the middle tier if it supports a NatStar ORB engine -- AIX Oracle (for instance).

7.4 Restrictions

When you use ORACLE, under TDS, you must be carefull about the connect on the data base, you must respect the following rules:

1. For a transaction there is only one data base which is committed:
   - it is the first one which asks the commit (the following one which ask the commit are rejected),
   - or it is the last data base successfully reached the other ones are automatically rollbacked.
2. To perform correctly the access to the data on a single NatStar transaction, you may simultaneously:
   - update only ONE data base and
   - read several data base.

   You may not authorized to update several data base simultaneously.
3. To connect several data base you must choice a different DB name in each SQL_OPEN. If you choice the same name then the last SQL_OPEN is disconnected and the new one is connected, all update are forgotten except if a commit of the data base has been explicitly requested.

8. How to Use the NSCOM Light (NS02OP7G)

8.1 Getting Started

The communication driver NS02OP7G is a BULL proprietary product.

The NS02OP7G increases the communication performance as compared to NSCOM (NS02COMT). For example, a remote function that reads 1 record on a UFAS indexed file (send data + receive data on NATSTAR client) is performed in 1.27 seconds instead of 0.33 seconds with NS02OP7G.

The delivery is made via an add-on diskette.

To use the NS02OP7G driver:

1. From the diskette, do:

   ```
   A:INSTALL <full path of NATSTAR installation>
   ```

   from the menu START -> EXECUTE...

2. Add the service in Options -> Services - Libraries: the NS02OP7G services.

3. In NATSTAR change:

   ```
   Options -> Configuration (configuration client) -> Misc -> IM :
   Transaction Monitor : NS02OP7G (instead of NS02COMT)
   ```

4. Import the connection/de-connection template from G7_LINK located in:

   ```
   %NATSTAR%\gcos\exp\gcos7lnk.exp export file
   ```

5. Add the template connection/de-connection template G7_LINK in the client application. If you have some problems to insert this template, then copy and paste this template into your application. Note that this template is an example of connection/de-connection, and you can see the associated code via a push button, to produce you specific needs.

6. Run open7 gateway under open7 and under root using the command:

   ```
   /users/op7gtw/op7gw2 -p 9002
   ```

   if op7gw2 is not already running (9002 represents the N° of port that it will be necessary to indicate in the virtual node defined in the connection template).

7. To stop open7 gateway, execute the following under root:

   ```
   /users/op7gtw/op7gw2Stp
   ```

8.2 The NS02OP7G API

There are 3 new APIs for NS02OP7G:

- NSTC_DEFINE_VIRTUAL_NODE
- NSTC_CONNECT
- NSTC_DISCONNECT

8.2.1 NSTC_DEFINE_VIRTUAL_NODE

Prototype:

```
FUNCTION NSTC_DEFINE_VIRTUAL_NODE
                                    (CSTRING VIRTUAL_NODE,
                                     CSTRING USER,
                                     CSTRING PASSWORD,
                                     CSTRING OPEN7_NAME,
                                     CSTRING PORT_NUM,
                                     CSTRING GCOS7_NAME,
                                     CSTRING TDS_NAME,
                                     CSTRING PROJECT,
                                     CSTRING BILLING,
                                     CSTRING TRANSACTION_NAME)
                                     RETURN INT(4)
```

Purpose:

Allows you to define a virtual node. A virtual node is a set of data which identifies a DSA connection for GCOS 7/TDS, and a TPCI/IP data connection for OPEN 7. A virtual node is also the logical name of the server used in the THINGS_SET_DEFAULT_SERVER instruction.

You can redefine a virtual node by using the same VIRTUAL_NODE and overwriting with a different value. If you specify an empty string for a parameter, than the last value of the corresponding parameter is kept except for the parameters which can be empty (in this case, you must give the full new value).

You can avoid defining the virtual node if you use the DSNCNFG utility to create a file containing all the virtual nodes that you need. This file is like the nscom.ini file, but you cannot edit it using a text editor but by using the DSNCNFG utility (because the created file is in binary mode for reasons of confidentiality, due to the encode of the password).

Parameters:

Input:

CSTRING VIRTUAL_NODE:
is the name of the virtual name that you are actually define.
Size: 12 characters maximum.

CSTRING USER: is the user name known by GCOS 7 for TDS connection
Size: 12 characters maximum.

CSTRING PASSWORD: is the password of the user (it can be empty)
Size: 12 characters maximum.

CSTRING OPEN7_NAME:
is the name of the OPEN 7 (this name must be entered in the hosts file on the PC) on which the OPEN7 GATEWAY is running.
Size: 8 characters maximum.

CSTRING PORT_NUM: is the port number on which the OPEN7 GATEWAY is waiting. The usual value is 9002.
Size: 8 characters maximum.

CSTRING GCOS7_NAME:
is the name of GCOS 7 host.
Size: 4 characters maximum.

CSTRING TDS_NAME: is the name of the TDS (or TDS mailbox) on which the NATSTAR transactions are loaded.
Size: 4 characters maximum.

CSTRING PROJECT: is the project name of the user (it can be empty).
Size: 12 characters maximum.

CSTRING BILLING: is the billing name of the user (it can be empty).
Size: 12 characters maximum.

CSTRING TRANSACTION_NAME:
is the name of transaction to be executed.
Size: 8 characters maximum.

Output:

INT(4): is the result of the function. It's value is NST_OK if the function is successfully completed, otherwise, the result may be interpreted by the NSTM_ERROR_MSG function.

Remarks:

To allow the execution of different transactions on the same session (or username), you can define several virtual nodes with the same value (except the TRANSACTION_NAME). Then, you establish a connection with a virtual node, and you can execute another transaction on this session, if you select another virtual node (like the first one, except for the transaction name) by the THINGS_SET_DEFAULT_SERVER instruction.

8.2.2 NSTC_CONNECT

Prototype:

```
FUNCTION NSTC_CONNECT(CSTRING SERVICE) RETURN INT(4)
```

Purpose:

Allows you to open a session between NATSTAR on PC and TDS on GCOS 7 through OPEN7 GATEWAY. This virtual node is also the logical name of the server used in the THINGS_SET_DEFAULT_SERVER instruction.

Parameter:

Input:

CSTRING SERVICE: is the name of a virtual node.

Output:

INT(4):	is the result of the function. It's value is NST_OK, if the function is successfully completed, otherwise, the result may be interpreted by the NSTM_ERROR_MSG function.

8.2.3 NSTC_DISCONNECT

Prototype:

```
FUNCTION NSTC_DISCONNECT(CSTRING SERVICE) RETURN INT(4)
```

Purpose:

Allows you to close a session between NATSTAR on PC and TDS on GCOS 7 through OPEN7 GATEWAY. This virtual node is also the logical name of the server used in the THINGS_SET_DEFAULT_SERVER instruction.

Parameters:

Input:

CSTRING SERVICE:	is the name of a virtual node.

Output:

INT(4):	is the result of the function. It's value is NST_OK, if the function is successfully completed; otherwise, the result may be interpreted by the NSTM_ERROR_MSG function.

9. How to Use the COBOL Subroutines From the NatStar NCL Language

9.1 Method

The method used here (inclusion code with %) is easy to use, but if you want to take care of the NCL (NatStar Control Language) portable code, you must not use this method.

The portable method is described in the NATSYSTEM documentation: *NATSTAR Reference of NCL Language*, paragraph *FUNCTION EXTERNAL and INSTRUCTION EXTERNAL*.

You can install the following example programs to check the compatibility of the other types of parameters. For this, refer to the NATSYSTEM documentation: *NATSTAR Reference of NCL Language*, section: *Link between C Language and PASCAL*.

In the NATSTAR GCOS 7 documentation, you must read the following sections:

- 5.3.2 *Using Packed-Decimal Data Type in NCL*
- 5.3.4 *Using Native Code*
- 5.3.5 *Calling a Conventional Subroutine*

This is an example of 2 COBOL programs: 1 main and 1 subroutine:

1. The 1st: DISPLAYMAIN is a COBOL main program which call a COBOL subroutine
2. The 2nd: DISPLAYPGM is a COBOL subroutine.

After the description of these programs, the following are presented:

- the COBOL compilation,
- linker, and
- execution of the DISPLAYMAIN.

Now, the following example shows how an NCL procedure can call a COBOL subroutine.

To produce a batch program instead of a TPR, we choose a configuration to generate a BATCH program. This choice does not affect this demonstration, and you may re-use it to test some parameter types to be passed to the subroutine.

9.2 Examples

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. DISPLAYMAIN.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. DPS7.
 OBJECT-COMPUTER. DPS7.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 W-INT2      COMP-1    VALUE 127.
 77 W-INT4      COMP-2    VALUE 70000.
 77 W-STRING    PIC X(12) VALUE "Test du disp".
 PROCEDURE DIVISION.
 DEBUT.
      CALL "DISPLAYPGM" USING W-INT2, W-INT4, W-STRING.
      STOP RUN.
 END PROGRAM DISPLAYMAIN.
-----------------------------------------------------------------
 IDENTIFICATION DIVISION.
 PROGRAM-ID. DISPLAYPGM.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. DPS7.
 OBJECT-COMPUTER. DPS7.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 LINKAGE SECTION.
 77 P-INT2      COMP-1.
 77 P-INT4      COMP-2.
 77 P-STRING    PIC X(12).
 PROCEDURE DIVISION USING P-INT2, P-INT4, P-STRING.
 DEBUT.
      DISPLAY "P-INT2=", P-INT2, ", P-INT4=", P-INT4, ",
               P-STRING=", P-STRING UPON TERMINAL.
      EXIT PROGRAM.
 END PROGRAM DISPLAYPGM.
-----------------------------------------------------------------
S: lmn sl rpc.z1.sllib,command='move DISPLAYMAIN type=cbx; move
DISPLAYPGM type=cbx;'
>>>11:49 LMN  70.00    27 -11
C: MOVE DISPLAYMAIN TYPE=CBX;
```

```
C: MOVE DISPLAYPGM TYPE=CBX;
<<<11:49
S: COBOL SOURCE=DISPLAYMAIN LIB=RPC.Z1.SLLIB,
XREF,MAP,NCKSEQ,OBJLIST,CODAPND,NCASEQ,NCARDID,LEVEL=NSTD;
>>>11:49 COBOL V-05.17-2
   MAR 19, 1998 11:49:12 X121.11 PROCESSING OF
           DISPLAYMAIN:RPC.Z1.SLLIB
   SUMMARY FOR DISPLAYMAIN: NO ERRORS, NSTD=2 CU PRODUCED.
<<<11:49
S: COBOL SOURCE=DISPLAYPGM LIB=RPC.Z1.SLLIB,
XREF,MAP,NCKSEQ,OBJLIST,CODAPND,NCASEQ,NCARDID,LEVEL=NSTD;
>>>11:49 COBOL V-05.17-2
   MAR 19, 1998 11:49:48 X121.12 PROCESSING OF
     DISPLAYPGM:RPC.Z1.SLLIB
   SUMMARY FOR DISPLAYPGM: NO ERRORS, NSTD=3 CU PRODUCED.
<<<11:49
S: lk DISPLAYMAIN
>>>11:55 LK  120.0    31  -4
   WORKING ON: DISPLAYMAIN
   LK00.(120.00)  SUMMARY FOR  DISPLAYMAIN  NO ERROR DETECTED .
    OUTPUT MODULE PRODUCED
<<<11:55
S: exec_pg DISPLAYMAIN
   P-INT2=   127, P-INT4=      70000, P-STRING=Test du disp
S:
====================================================================
====================================================================
WITH NATSTAR ...
------------

This is an EXPORT file:

@MODE GRAMMAR
; File exported by NATSYS on 20/03/98,14:23:55
; File exported with IMEX_IMEX.DLL dated on Nov 12 1997
   version 1080
; Resources exported:
;     L.GCOS7_CALL
; Options:
;     EXTERNALS=ON
;     DESCRIPTIONS=ON
;     DOCS=ON
;     ALIAS AND KEYWORDS=ON
;     USES=OFF
;     ZEROSIZED=ON
;     CLASS WITH METHODDEF=OFF
;     WITH METHOD IMPLEMENTATION=ON
;     ONLY ON KEYWORD=OFF
;
;----------------------- Beginning of Library GCOS7_CALL
-----------------------
```

```
LIBRARY GCOS7_CALL
        DESCR = 'Example to call COBOL S/P in NCL. '
        DOC 0
        ENDDOC
        FUNCTION NCL_DISPLAY_MAIN
                RETURN = INT(4)
                DOC 0
                ENDDOC
                PARAMETERS 0
                ENDPARAMETERS
                IMPLEM 15
local int    P1 (2)
LOCAL int    P2 (4)
LOCAL char   P3(12)

P1 = 32767
P2 = 77000
P3 = "From NCL pgm"

%
* Call the native COBOL sub-program:
     CALL "DISPLAYPGM" USING P1, P2, P3.
         STOP RUN.
 END-PGM.
%

                ENDIMPLEM
        ENDFUNCTION
ENDLIBRARY
;-------------------------- End of Library GCOS7_CALL
--------------------------
;*** END OF SECTION *** Lines written: 31,
                       Elapsed time: 0 seconds.
************


=====================================================
```

Note that the insertion after the call:

```
          STOP RUN.
 END-PGM.
```

must be done in case of a BATCH program, because the generated COBOL program ends with an EXIT PROGRAM, and the COBOL Run-Time-Package aborts when the main program execution terminates with an EXIT PROGRAM instead of a STOP RUN.

If the STOP RUN is not inserted then the following message is produced by the Run-Time-Package to abort the program:

```
   FATAL     EX03.     UNEXPECTED RETURN CODE (RC=F0001889->USER0,SEQERR)
                       GOT IN TASK MAIN.0  AT ADDRESS 8.10.00FE
   11.47 *** TASK MAIN PGID=75 PRID=0 ABORTED BY SYSTEM
                       RC=F0001889->USER   0,SEQERR
S:
=====================================================
```

The following configuration is used to produce a BATCH program from NCL

:

```
+-------------------------------------------+
|          \ Configuration NAME +            +
|Property   \                   + CF_BATCH   +
|-------------------------------+------------+
|MAIN                           +            +
|    * Kind                     +            +
|       Client                  +            +
|       Server                  +  *         +
|       Library                 +            +
|       Main Window             +            +
|    * OPtion                   +            +
|      Extended Ress            +            +
|-------------------------------+------------+
|TARGET                         +            +
|      LG Target                + COBOL      +
|      DB Target                + LOCAL      +
|      UI Target                + GCOS7      +
|-------------------------------+------------+
|IM                             +            +
|  Error Popup                  + *          +
|  Generate Database initializ  +            +
|  Static SQL mode generation   +            +
|  Transaction Monitor          +            +
|  Parameters                   + NAME=CLIENT +
|-------------------------------+------------+
|PM                             +            +
|  Generate initializ           +            +
|  Generate Multiple scenarii   +            +
|  Scenarii filename            + C:\...     +
+-------------------------------+------------+
```

After you build resources for L.GCOS7_CALL library, you can produce the COBOL by means of COBGEN.

The generated COBOL is n_gcosck.cbl:

```
==========================================================

************************************************************
*                                                          *
* module N_GCOSCK                                          *
* Program generated with Nat Systemes Code Generator       *
* 1.00.000                                                 *
* Copyright (C) 1996 Nat Systemes. All Rights Reserved.    *
*                                                          *
************************************************************
```

```
 IDENTIFICATION DIVISION.
     COPY NSCOBOL.CPY.
     COPY TERMREPL.CPY.


*                                                                *
*  Function _GCOSCK__DISPATCH%                                   *
*                                                                *
 PROGRAM-ID. IZGCOSCKZZDISPATCH.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
::NS-SOURCE-COMPUTER::
::NS-OBJECT-COMPUTER::
 SPECIAL-NAMES.
::NS-CALL-CONVENTION::

 DATA DIVISION.
*                                                                *
* _GCOSCK__DISPATCH%   VARIABLES AND TEMPORARIES                 *
*                                                                *
 WORKING-STORAGE SECTION.
 COPY NZGCOSCK.CPY.
 01 I-STL::NS-POINTER:: .
 01 TMP-COBOL-GEN-0000::NS-SHORT::.
 01 TMP-COBOL-GEN-0001::NS-SHORT::.
 01 TMP-COBOL-GEN-0002::NS-SHORT::.
 01 TMP-COBOL-GEN-0003::NS-LONG::.

 LINKAGE SECTION.
 01 TMP-RESULT::NS-LONG::.
 01 PP-TMP-RESULT::NS-POINTER:: .
 01 SZ-PP-SERVICE::NS-POINTER:: .
 01 SZ-SERVICE PIC X(32) USAGE DISPLAY.
 01 SZ-R-SERVICE REDEFINES SZ-SERVICE.
 02 SZ-ACH-R-SERVICE PIC X OCCURS 32.
 01 I-PP-STL::NS-POINTER:: .
 01 I-PAR-STL::NS-POINTER:: .
 PROCEDURE DIVISION::NS-CALL:: USING PP-TMP-RESULT SZ-PP-SERVICE
      I-PP-STL.
 PROG.

     SET ADDRESS OF TMP-RESULT TO PP-TMP-RESULT
     SET ADDRESS OF SZ-SERVICE TO SZ-PP-SERVICE
     SET ADDRESS OF I-PAR-STL TO I-PP-STL
     SET I-STL TO I-PAR-STL

     MOVE 1 TO TMP-COBOL-GEN-0000
     MOVE 2 TO TMP-COBOL-GEN-0001
     MOVE 14 TO TMP-COBOL-GEN-0002
     MOVE 168 TO TMP-COBOL-GEN-0003
     CALL::NS-CALL:: "CBL0THINGS_SET_ERROR"  USING BY REFERENCE
```

```
     ADDRESS OF TMP-COBOL-GEN-0000
     BY REFERENCE ADDRESS OF TMP-COBOL-GEN-0001
     BY REFERENCE ADDRESS OF TMP-COBOL-GEN-0002
     BY REFERENCE ADDRESS OF TMP-COBOL-GEN-0003
     BY REFERENCE ADDRESS OF SZ-SERVICE
     MOVE -1 TO TMP-RESULT
     EXIT PROGRAM
     MOVE 0 TO TMP-RESULT
     EXIT PROGRAM
     .


 END PROGRAM IZGCOSCKZZDISPATCH.


*                                                                *
*  Function NCL_DISPLAY_MAIN                                     *
*                                                                *
 IDENTIFICATION DIVISION.
     COPY NSCOBOL.CPY.
     COPY TERMREPL.CPY.
 PROGRAM-ID. NCLZDISPLAYZMAIN.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
::NS-SOURCE-COMPUTER::
::NS-OBJECT-COMPUTER::
 SPECIAL-NAMES.
::NS-CALL-CONVENTION::

 DATA DIVISION.
*                                                                *
* NCL_DISPLAY_MAIN   VARIABLES AND TEMPORARIES                   *
*                                                                *
 WORKING-STORAGE SECTION.
 COPY NZGCOSCK.CPY.
 01 P1::NS-SHORT::.
 01 P2::NS-LONG::.
 01 P3 PIC X(12) USAGE DISPLAY.
 01 R-P3 REDEFINES P3.
 02 ACH-R-P3 PIC X OCCURS 12.
 01 TMP-COBOL-GEN-0004::NS-SHORT::.

 LINKAGE SECTION.
 01 TMP-RESULT::NS-LONG::.
 01 PP-TMP-RESULT::NS-POINTER:: .
 PROCEDURE DIVISION::NS-CALL:: USING PP-TMP-RESULT.
 PROG.

     SET ADDRESS OF TMP-RESULT TO PP-TMP-RESULT
```

```
     MOVE 32767 TO TMP-COBOL-GEN-0004
     COMPUTE P1 ROUNDED = TMP-COBOL-GEN-0004
     MOVE 77000 TO P2
     MOVE "From NCL pgm" TO P3

*  START USER CODE SEQUENCE                                    *

* Call the native COBOL sub-program:
     CALL "DISPLAYPGM" USING P1, P2, P3.
         STOP RUN.
 END-PGM.

*  END USER CODE SEQUENCE                                      *
     MOVE 0 TO TMP-RESULT
     EXIT PROGRAM
     .


 END PROGRAM NCLZDISPLAYZMAIN.


=========================================================

The Copy COBOL is nzgcosck.cpy:

=========================================================

*                                                            *
*  Global Variable(s) structure include file.                *
*                                                            *

 01 NzGCOSCKzGLOBAL EXTERNAL.
 02 I---CL--GCOSCK::NS-POINTER:: .
 02 I---NSGLV-73-TAB::NS-POINTER:: .
 02 I---NSGLM-73-TAB::NS-POINTER:: .
=========================================================
```

The compilation of n_gcosck.cbl is aborted because, in the case of BATCH configuration, NATSTAR generates the IZGCOSCKZZDISPATCH program which is unused in this case.

You can use the EDITION_BATCH editor program with NSCOB7 **only** if you want to test a COBOL subroutine with a BATCH generated by NATSTAR.

The editor program for batch is (and it must be located in):

```
SI7.NS7.SLLIB..EDITION_BATCH
=====================================================
EDIT;
">>> EDITION_BATCH V1: 14.01.98 For Cobgen.exe 2.00.030
YV
B4
A
B0
^N
"-------------Transform the name of the COPY COBOL <name>.CPY into
<name>-CPY.
GS/#C.CPY/-CPY/
"-------------The main entry point is &2:
"             The H_CLR_EPROLOG and H_CLR_EPILOG is not CALLED.
"             But H_CLR_EPROLOG will be called in the first call
"             of the RTP function but the H_CLR_EPILOG will
"             not be called and some function as flush must
"             be called before the completion of this program.
"  Note that: there is a parameter which is called ``tmp-result''
"              this definition map the OPTION-STRING parameter.
^/PROGRAM-ID. *#(2)/
/ PROCEDURE DIVISION/
.,/END PROGRAM /GD/ TMP-RESULT/
[F
GS/#(2)/&2/
GS/#/[/
B0
R IL1: &1
GS/[X0D//
[B4
Z(CBX) &1
Q
=====================================================


S: lmn sl rpc.z1.sllib,command='move n_gcosck_cbl type=cbx; move
nzgcosck_cpy new=nzgcosck-cpy type=cbx rpl;move
DISPLAYMAIN type=cbx; move DISPLAYPGM type=cbx;'
>>>14:02 LMN  70.00    27 -11
C: MOVE N_GCOSCK_CBL TYPE=CBX;
C: MOVE NZGCOSCK_CPY NEW=NZGCOSCK-CPY TYPE=CBX RPL;
C: MOVE DISPLAYMAIN TYPE=CBX;
C: MOVE DISPLAYPGM TYPE=CBX;
<<<14:02
S: ejr NSCOB7,,NS.TEST.SLLIB vl=(n_gcosck_cbl, MODE=BATCH,
MAIN=NCLZDISPLAYZMAIN, INLIB=RPC.Z1.SLLIB, CULIB=NS.TEST.CULIB,
NSSLLIB=NS.TEST.SLLIB, TMPLIB1=COMMON.SLLIB,PRTLIB=RPC.Z1.LISLIB)
holdout;p
>>14:03 LISTEN.
```

```
--> BREAK TO RETURN TO INT MODE
-->14.03 X321  IN   NSCOB7 USER=RPC_3 CLASS=P  SPR=7 STATION=BC04
-->14.03 X321     STARTED   NSCOB7   RPC_3 P
-->  FROM   RPC_3 X321: >>> START NSCOB7 FOR N_GCOSCK_CBL: ,FRM,
-->  FROM   RPC_3 X321: >>>              MODE=BATCH
-->  FROM   RPC_3 X321: >>>              INLIB
=RPC.Z1.SLLIB::..N_GCOSCK_CBL
-->  FROM   RPC_3 X321: >>>              TMPLIB1=COMMON.SLLIB,
-->  FROM   RPC_3 X321: >>>              PRTLIB =RPC.Z1.LISLIB::,
-->  FROM   RPC_3 X321: >>>              CULIB  =NS.TEST.CULIB::,
-->  FROM   RPC_3 X321: >>>              NSSLLIB=NS.TEST.SLLIB::.
-->  FROM   RPC_3 X321: >> EDITION Input
=RPC.Z1.SLLIB::..N_GCOSCK_CBL.
-->  FROM   RPC_3 X321: >> (BATCH)
Output=COMMON.SLLIB..N_GCOSCK_CBL.
-->  FROM   RPC_3 X321: >>         MAIN  =NCLZDISPLAYZMAIN.
-->  FROM   RPC_3 X321: >>         Exec
=NS.TEST.SLLIB::..EDITION_BATCH.
-->  FROM   RPC_3 X321: >> COMPILE-COBOL
Source=COMMON.SLLIB..N_GCOSCK_CBL.
-->  FROM   RPC_3 X321: >>              Inlib1=RPC.Z1.SLLIB::.
-->  FROM   RPC_3 X321: >>              Inlib2=NS.TEST.SLLIB::.
-->  FROM   RPC_3 X321: >>              Prtlib=RPC.Z1.LISLIB::..*
-->  FROM   RPC_3 X321: >>              Culib =NS.TEST.CULIB::.
-->  FROM   RPC_3 X321: ==> ERROR(S) IN COMPILE COBOL FOR
                                            N_GCOSCK_CBL.
-->14.03 X321.3    ABORTED   NSCOB7   RPC_3 P SEV3 $*$BRK


S: scn
>>>14:05 SCANNER  70.01
C: pr X321:1
   X321:1
   LINES:     264   PAGES: unknown
...
      >> COMPILE-COBOL Source=COMMON.SLLIB..N_GCOSCK_CBL.
      >>               Inlib1=RPC.Z1.SLLIB::.
      >>               Inlib2=NS.TEST.SLLIB::.
      >>               Prtlib=RPC.Z1.LISLIB::..*
      >>               Culib =NS.TEST.CULIB::.
      STEP 3 COBOL
      LOAD MODULE = H_COBOL85 (11:16 DEC 04, 1997 )PREINITIALIZED
      LIBRARY = SYS.HLMLIB
WARNING   DY02. ERROR DETECTED: SHARED MODULE H_SM14 NOT FOUND
14:03:13  STEP STARTED XPRTY=9 (MAR 20, 1998)
          CBL02. SUMMARY FOR IZGCOSCKZZDISPATCH: ***=1 **=1 *=182
NSTD=22 NO CU PRODUCED.
          CBL02. SUMMARY FOR NCLZDISPLAYZMAIN: **=1 *=182 NSTD=8
          CU PRODUCED.
          TASK MAIN PGID=0008 PRID=00 COMPLETED
          RC TRACE:   RC=4E901008->QL    16,SFNUNKN  AT
```

```
1E8A074C             RC=4E901008->QL    16,SFNUNKN  AT 1E8A074C
                       RC=4E901008->QL    16,SFNUNKN  AT
1E8A074C             RC=4E990440->QL    25,WRONGITM AT 1E8A96A4

          SYSBKST   ON  BFS1P1:  NB  OF  IO  REQUESTS=          0
          SYSPVMF   ON  ******:  NB  OF  IO  REQUESTS=          2
          SYSLIB    ON  ******:  NB  OF  IO  REQUESTS=        159
          SYSBKST*  ON  ******:  NB  OF  IO  REQUESTS=        157
          H_INLIB1  ON  BFU428:  NB  OF  IO  REQUESTS=          8
          H_INLIB2  ON  BFU42A:  NB  OF  IO  REQUESTS=         14
          H_INLIB   ON  BFU423:  NB  OF  IO  REQUESTS=          6
          H_PR      ON  BFU428:  NB  OF  IO  REQUESTS=         58
          H_CULIB   ON  BFU423:  NB  OF  IO  REQUESTS=         14
          CPU         0.065                        PROG MISSING
PAGES      199   STACKOV       1
          ELAPSED     0.183                        SYS  MISSING
PAGES        0
          LINES            0   LIMIT     NOLIM      BACKING
STORE             0   LOCKED   143360
          CARDS            0   LIMIT     NOLIM      BUFFER SIZE
110592   CPSIZE     4096
14:03:24  STEP    ABORTED   SEV3         (MAR 20, 1998)
...

S: ejr NSLK7,,NS.TEST.SLLIB vl=(NCLZDISPLAYZMAIN, TYPE=LM,
LNKSL=NS.TEST.SLLIB, LINK=BATCH_LNK,  OUTLIB=NS.TEST.LMLIB,
PRTLIB=NS.TEST.SLLIB, CULIB=NS.TEST.CULIB, NSCULIB=NS.ADV.CULIB)
holdout;p

>>14:04 LISTEN.
--> BREAK TO RETURN TO INT MODE
-->14.04 X324  IN   NSLK7 USER=RPC_3 CLASS=P  SPR=7 STATION=BC04
-->14.04 X324     STARTED   NSLK7    RPC_3 P
-->      FROM   RPC_3 X324: >>> START NSLK7 FOR NCLZDISPLAYZMAIN
TYPE=LM, USE=NONE.
-->      FROM   RPC_3 X324: >> Prepare NS_BATCH_LNK Linker command
from NS.TEST.SLLIB::.
-->      FROM   RPC_3 X324: >> LINKER for NCLZDISPLAYZMAIN
INLIB=NS.TEST.CULIB::,
-->      FROM   RPC_3 X324: >>        Inlib1=NS.ADV.CULIB::,
-->      FROM   RPC_3 X324: >>        Outlib=NS.TEST.LMLIB::,
-->      FROM   RPC_3 X324: >>        Prtlib=NS.TEST.SLLIB::,
-->      FROM   RPC_3 X324: >>       Comfile=TEMP..NS_BATCH_LNK.
-->      FROM   RPC_3 X324: >>> END LKLM FOR NCLZDISPLAYZMAIN.
-->14.04 X324.2    COMPLETED NSLK7    RPC_3 P $*$BRK

S: exec_pg NCLZDISPLAYZMAIN lib=NS.TEST.LMLIB;
   P-INT2= 32767, P-INT4=      77000, P-STRING=From NCL pgm
S:
```

Now, you are ready to use a COBOL subroutine, the only problem that you can meet is the type of the parameters.

In this example, the types used are:

- COMP-1
- COMP-2
- PIC X(n)

A. Introduction to the NatStar Development Workstation

A.1 NatStar Development Workstation Session

The appendices B through F that follow guide you through a session on the NatStar development workstation. During this session you will develop a simple client/server application in which the server code will execute on GCOS 7. The client code will execute on the PC you use as the development workstation. For simplicity, this example assumes that you will use a Windows 95 PC as the NatStar development workstation.

The steps described in the example are illustrated using screen snapshots of the NatStar 2.10 software.

NOTE:

The example in these appendices assumes that you are generally familiar with the use of the NatStar development workstation, so the following is not a beginner's tutorial on how to develop client/server applications. Additional information about using the NatStar development workstation is provided in the Nat Systems documentation (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this manual).

Appendix B

Describes development of the basic CONCAT server application.

Appendix C

Describes development of the client application (user-interface window and supporting logic) using the CONCAT server application.

Appendix D

Describes GCOS 7 configurations that enable the DPS 7000 to be used as a NatStar server.

Appendix E

Describes partitioning of the application into client/server parts and discusses:

- the CONNECT and DISCONNECT phases
- access to UFAS files.

Appendix F

Describes generation of the COBOL-85 source program for the server logic.

NOTE:

Other operations required to complete the implementation of a NatStar/GCOS 7 client/server application are described in detail in Chapter 5 *Implementing Client/Server Applications*. The purpose of the following example is to familiarize you with the aspects of NatStar/GCOS 7 client/server development that use the NatStar development workstation.

A.2 The Example Application

The example used here is based on the TESTUFAS application delivered with the NatStar/GCOS 7 product, in export file format EXAMPLE.EXP. If you use these appendices as an exercise, performing each step in the sequence described, the result will effectively be a copy of the application example.

The application in the TESTUFAS is called the CONCAT application because it performs a simple concatenation function between two strings. The CONCAT application includes a window, which is the interface to the user of the application. The window has two input fields, for values to be entered by the user, plus an output field where the concatenated value is displayed. The window also includes one push button in the basic step, for the CONCAT operations.

The WINUFAS application is defined in two libraries:

- The basic library **SRVUFAS** contains one function that performs the basic concatenation. The contents of the SRVUFAS library will later become the server code that is exported to GCOS 7 to operate as a TPR under TDS.
- The library **TESTUFAS** will contain the main window (user interface) and the associated logic. The contents of the TESTUFAS library will become the client code to operate on the client PC.

B. Developing the CONCAT Server Application

This section describes how to create the library and define the logic that will be associated with the server (GCOS 7) part of the CONCAT client/server application.

Before a NatStar application can call procedures, the procedures must be defined (as functions or instructions) in a library. The name, parameters (input/output), and the parameter data types for each procedure must be defined. When procedures are defined in the NatStar Repository, they can be referenced in code written in NCL (Nat Systems Control Language).

IMPORTANT:

Server code, such as server code that will execute on GCOS 7, must be contained in separate libraries from client code that will execute on client systems. Server code cannot be exported to a remote system if it is in the same library as client code.

B.1 Creating the Server Library

To create a library, click on the **Libraries** icon. The **Libraries** window is displayed, listing all libraries that are currently defined (see below).

NOTE:

The convention used throughout this example is that the left mouse button is used for all operations unless the right mouse button is specifically noted.

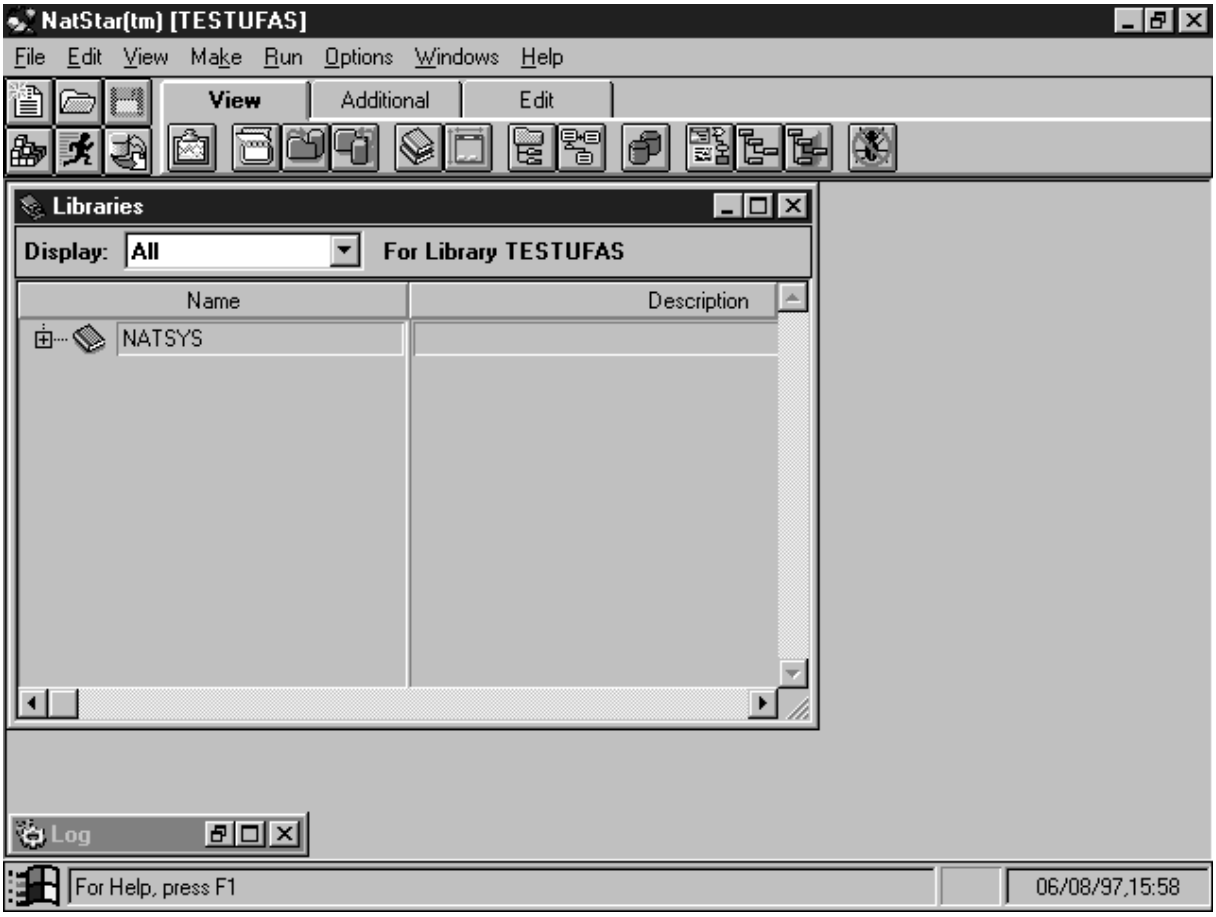

Figure B-1. Display Existing Libraries

To define the library that will contain the code to be executed on GCOS 7, select the **Insert Library** entry from the **Edit** menu (see below).

Figure B-2. Select the Insert Library Function

The **Insert New Library** window is displayed (see below).

In the **Name:** field, enter the name of the library -- **SRVUFAS** in this example -- and in the **Description:** field, enter a description of the library as shown in the figure.

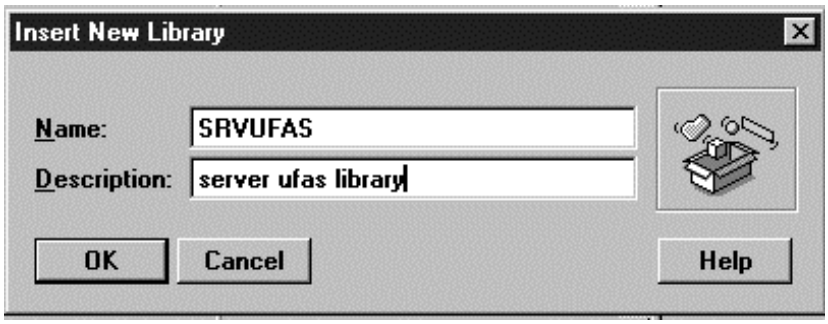

Figure B-3. Create the SRVUFAS Library

Click on the **OK** button to create the SRVUFAS library.

B.2 Defining Server Instructions

To associate resources -- that is, the application code -- with the library, you can begin by doing any of the following:

- press the **Insert** key on the keyboard, or
- select the **Insert...** entry from the **Edit** menu, or
- click on the **Insert** icon of the **Edit** tab.

The **Insert New Resource** window is displayed (see below).

Select **Function** from the **Kind** group of resources associated with the NatStar library being used to build the application. (To define a function, select **Function** from the **Kind** group.) Enter the name of the **SRV_CONC** function and a description of the function, as shown in the figure.

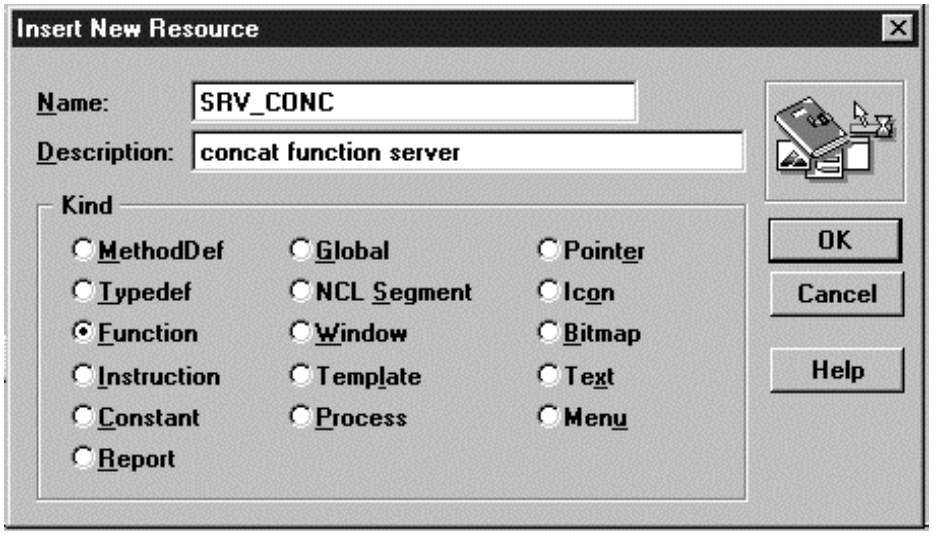

Figure B-4. Define a New Function

Click on **OK** to complete the definition of the new SRV_CONC function.

Double-click on the function name to invoke the **Modify function or instruction** window (see below). Select **Inline** so that you can enter the NCL to define the logic to be included in the instruction.

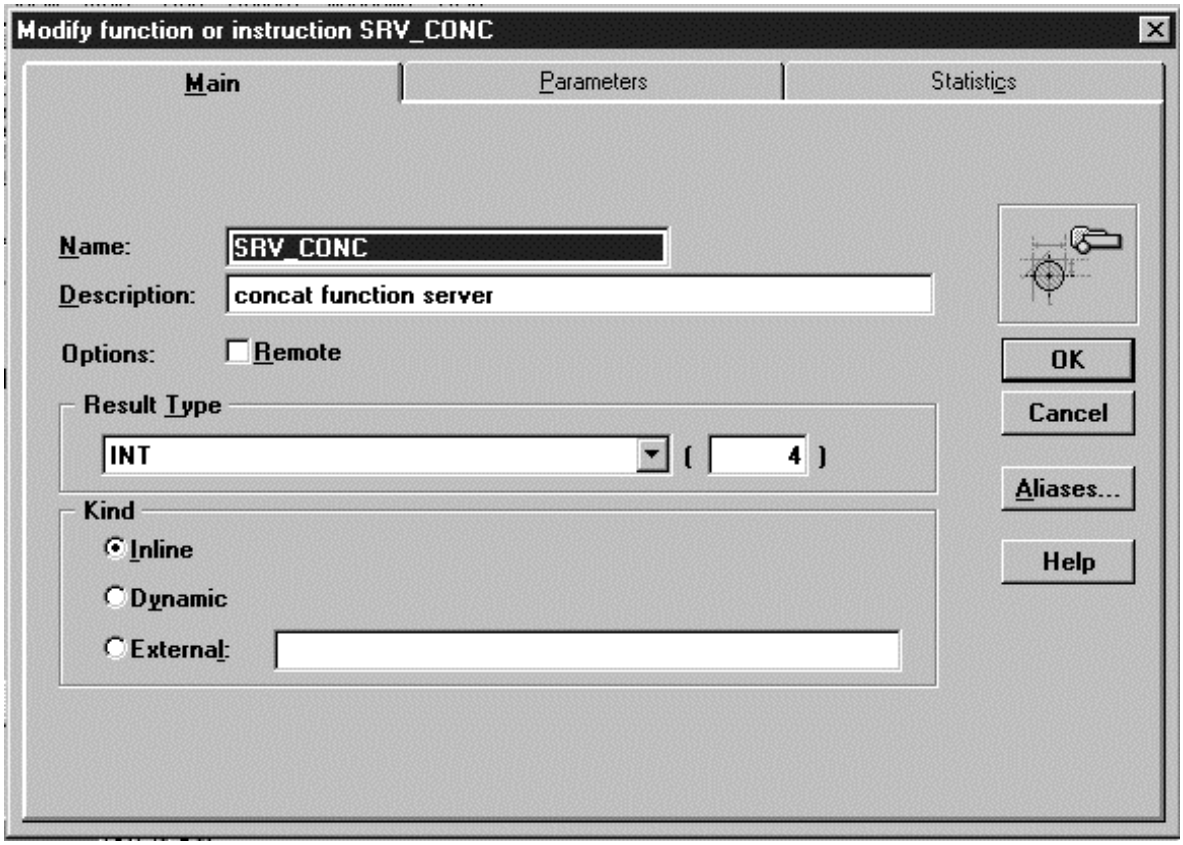

Figure B-5. Prepare to Enter NCL for the Instruction

Next, select the **Parameters** tab, which is used to define the parameters associated with the instruction (see below).

Define each parameter by entering its name, whether it is an input (**In**) or output (**Out**) parameter, and its data type, as shown in the figure. After you enter each parameter, click on the **Append** button to add the parameter to the parameter list.

NOTE:

The first two parameters are input parameters. The third parameter, which is shown highlighted in Figure B-6, is the output parameter.

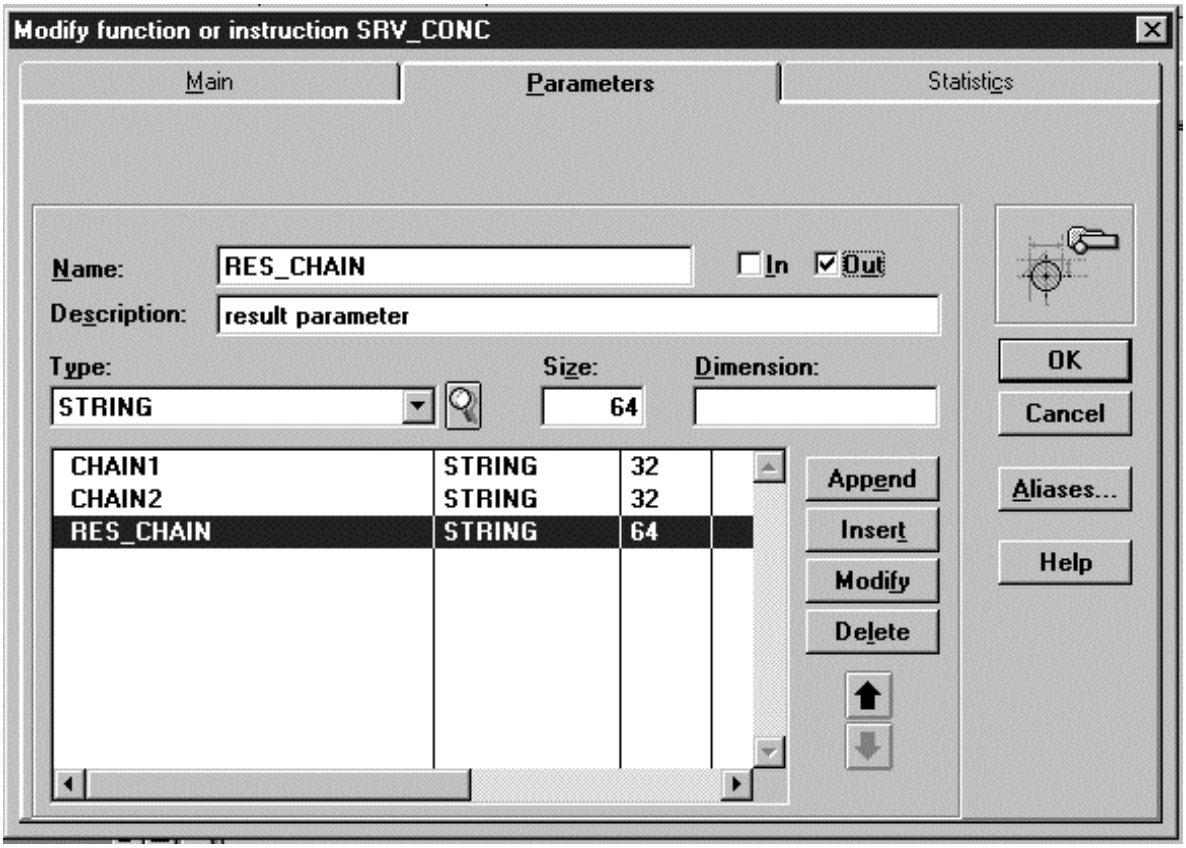

Figure B-6. Define the Instruction Parameters

After you have entered all of the parameters, click on **OK** to complete the definition of parameters.

You are now ready to enter NCL. To begin, double-click the right mouse button on the instruction name **SRV_CONC** under the library name **SRVUFAS**.

The **Edit** window of the **SRV_CONC** instruction is displayed, as shown below. Enter the NCL for the example application as shown in the figure. This NCL defines the logic that will be executed when the instruction is called.

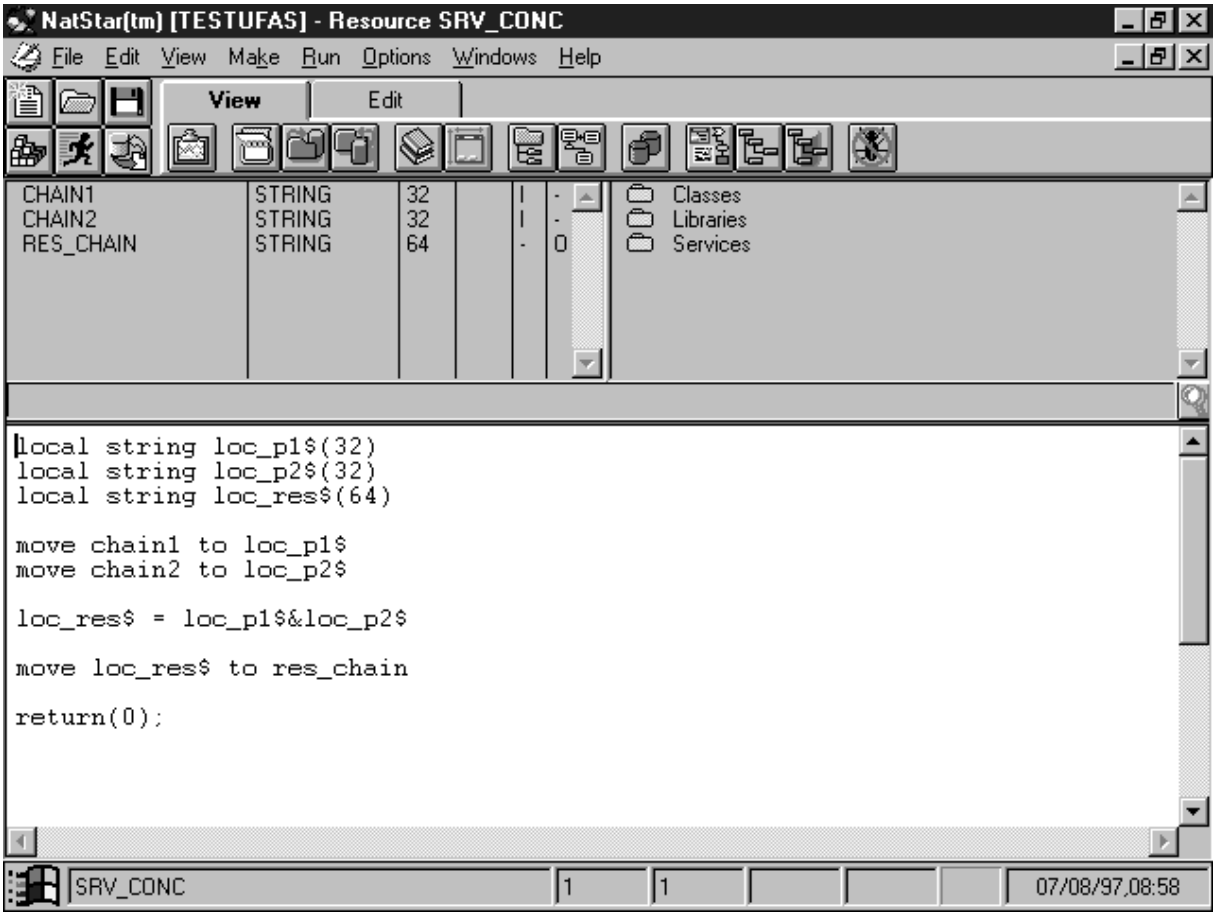

Figure B-7. Enter NCL for the SVR_CONC Instruction

For the full details of this screen, see Appendix G *NCL Sequence Listings*.

Verify that the NCL is correct by clicking on the **Check** icon of the EDIT tab. The **Check** icon activates a check of the syntax of the NCL. If necessary, refer to the *Nat Systems Documentation* for an explanation of errors that occur during syntax checking.

Any errors that occur during syntax checking appear in the status line of the display. If the NCL is correct, the function name, **SRV_CONC**, appears in the status line, as shown in the figure.

The return is mandatory in a function definition. It can be either a calculated value, or a state value (in which case the value is not significant).

Save the NCL by selecting **Save** from the **File** menu, or by clicking on the **Save** icon.

Close the **Resource SRV_CONC** window by selecting **Close** from the **File** menu.

C. Developing the Client Window and Logic

The next step to be performed consists in defining the client application. In this example, the client application consists of a single window with associated events. The user can enter two operands in the window and request that a concatenation function (CONCAT) be performed. The concatenation will be performed by the SRVUFAS server on GCOS 7 at a later stage, using the function just defined in appendix B. Firstly, the server will run on the local machine to execute the CONCAT function.

In this step, the client application is built. The client window will have two input operand fields and one result field. The window will have a push-button for the CONCAT operation.

C.1 Creating the Local Library

Remember that the logic to be executed on a remote server - GCOS 7 in this example - must be contained in a separate library from logic to be executed on the client. A library (called SRVUFAS) has already been created to contain the code to be executed on GCOS 7. Now it is necessary to create a local library to contain the client code.

To define the library that will contain the code to be executed on the client, select the **Insert Library** entry from the **Edit** menu (refer to Section B.1, *Creating the Server Library*).

The **Insert New Library** window is displayed (see the figure below).

In the **Name:** field, enter the name of the library -- **TESTUFAS** in this example -- and in the **Description:** field, enter the description shown.

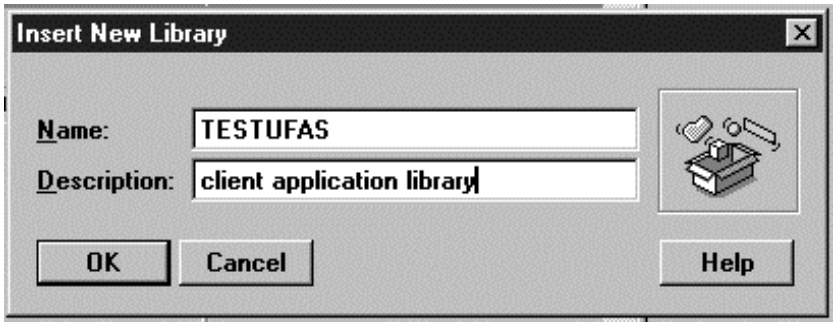

Figure C-1. Create the TESTUFAS Library

Click on **OK** to create the TESTUFAS library.

C.2 Defining the Client Window

To associate resources -- in this case the user-interface window -- with the TESTUFAS library, you can begin by doing any of the following:

- press the **Insert** key on the keyboard, or
- select the **Insert...** entry from the **Edit** menu, or
- click on the **Insert** icon of the **Edit** tab.

The **Insert New Resource** window is displayed (see the figure below). Select **Window** from the **Kind** group of resources. In the **Name:** field, enter **WINUFAS** as the name of the window, and enter a description of the window as shown in the figure.

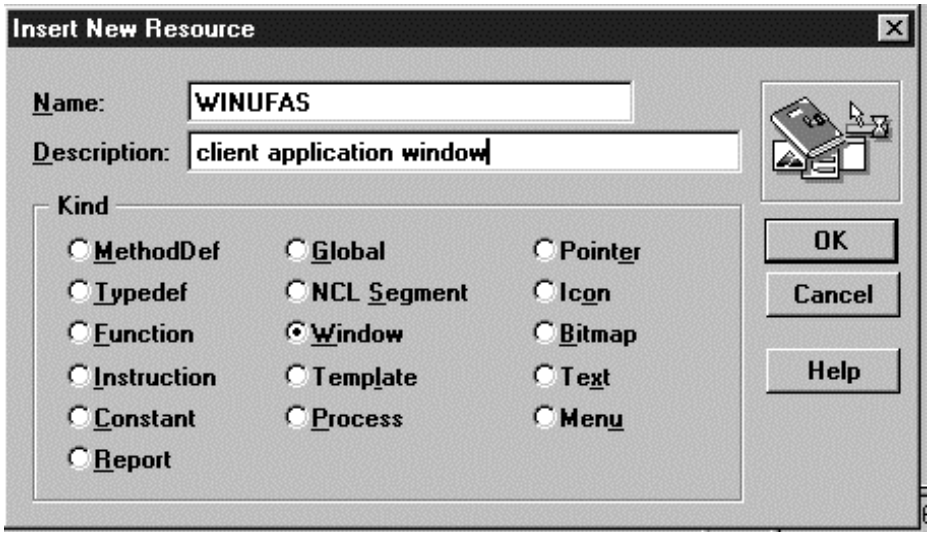

Figure C-2. Define the Client Window, Part 1

Click on **OK** to define the WINUFAS client window.

The **New Window or Template** window is displayed (see the figure below). Click on the dialog window icon. The dialog window icon is the second from the left in the top line of icons in the screen shown below, and is highlighted.

The **Model:** field will display the value **DIALOG**.

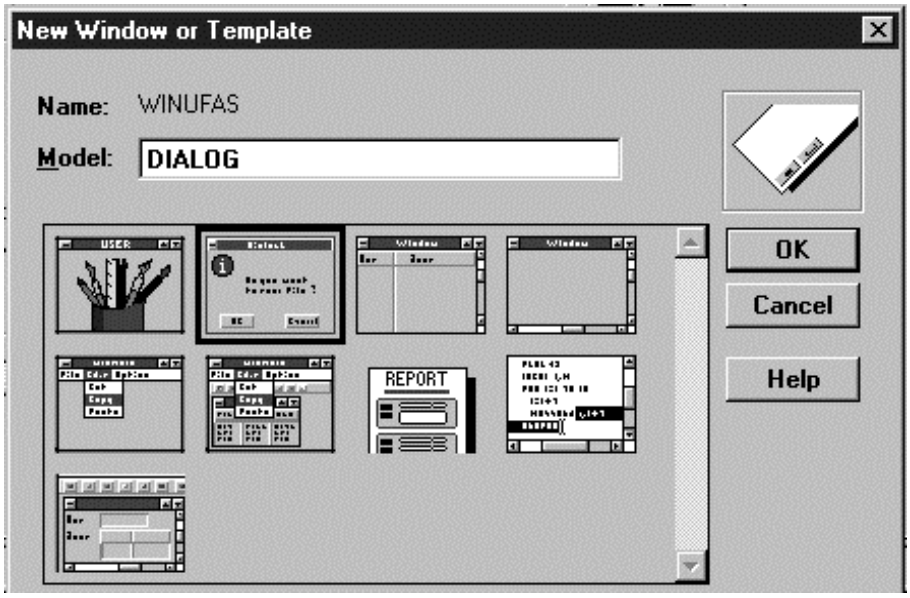

Figure C-3. Define the Client Window, Part 2

Click on **OK** to define the new window.

When you have completed the previous tasks, your **Libraries** window should resemble the window shown below.

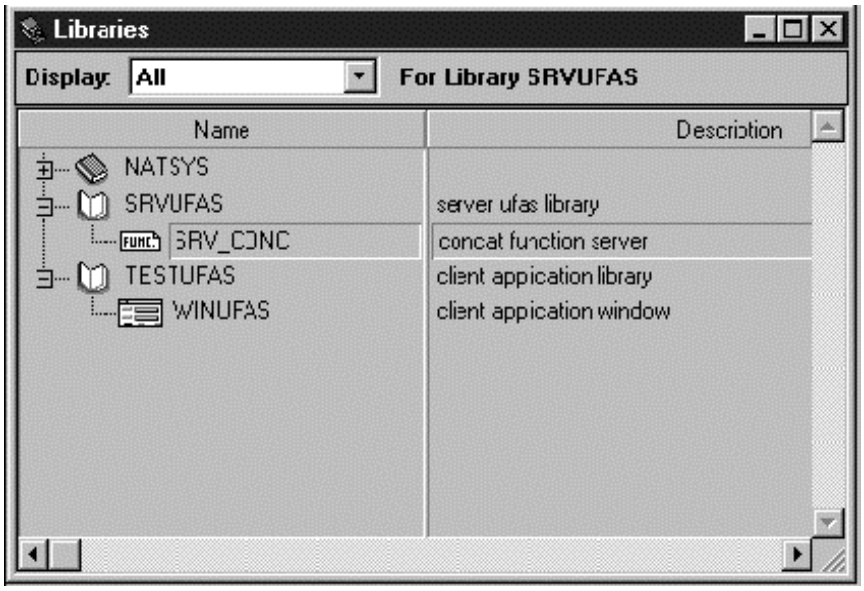

Figure C-4. Prepare to Define the User Interface

To define the user interface, which will be made up of entry fields, push-buttons, and text fields, double-click the right mouse button on the **WINUFAS** window entry icon in the library structure (shown above).

An empty version of the user interface window **WINUFAS** is displayed. To define the contents of the **WINUFAS** window, click on the **Controls** tab.

Select and drag the **Text** icon into the **WINUFAS** window. The **Control Name & Text** window is displayed (see below).

In the **Name:** field, enter **CHAIN1_TX**. In the **Text:** field, enter **string1**.

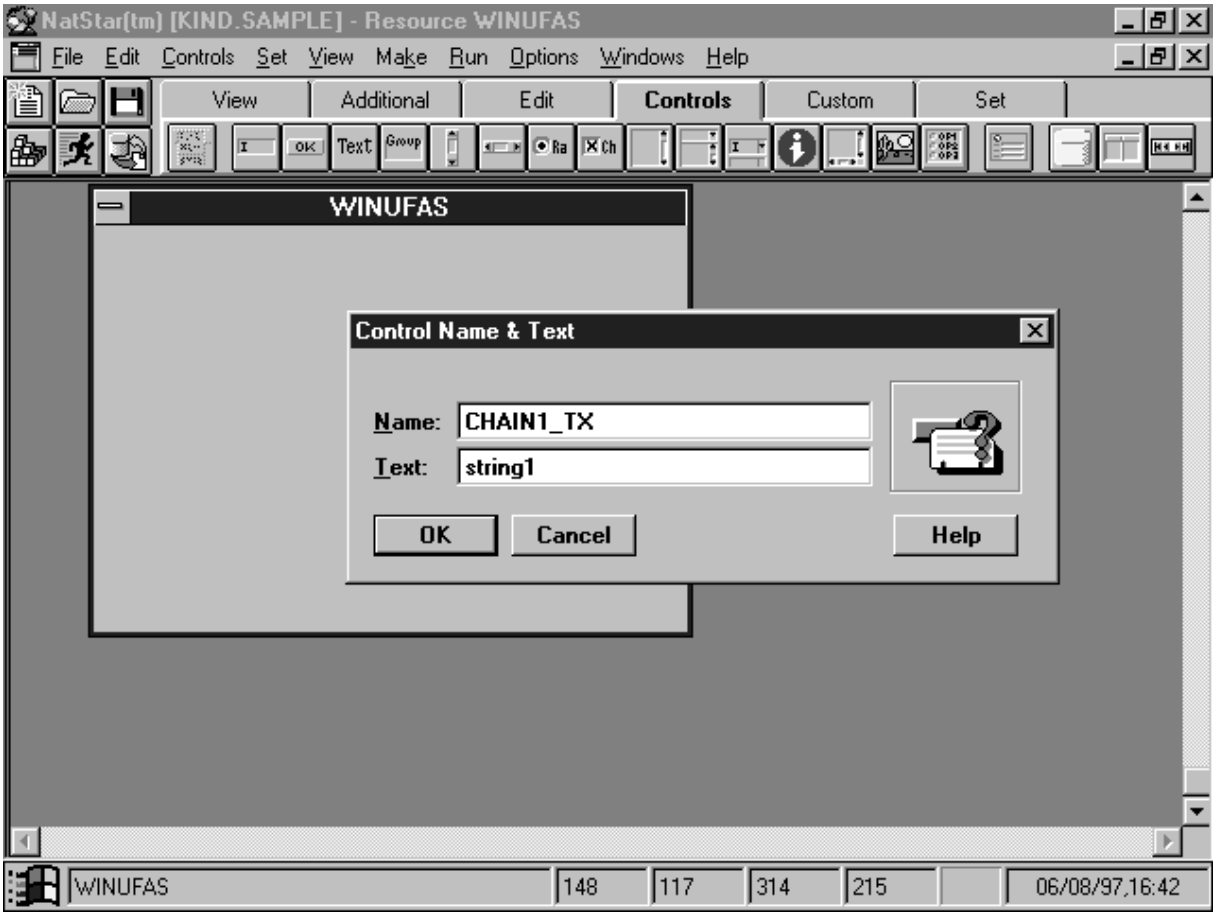
NatStar(tm) [KIND.SAMPLE] - Resource WINUFAS
File Edit Controls Set View Make Run Options Windows Help
View Additional Edit Controls Custom Set
WINUFAS
Control Name & Text
Name: CHAIN1_TX
Text: string1
OK Cancel Help
WINUFAS 148 117 314 215 06/08/97,16:42

Figure C-5. Define the Display Text for the First Operand

Click on **OK** to add the new control to the **WINUFAS** dialog window.

Next select and drag the **Entry** field icon into the **WINUFAS** window. The **Control Name** window is displayed (see below).

In the **Name:** field, enter **CHAIN1_EF**.

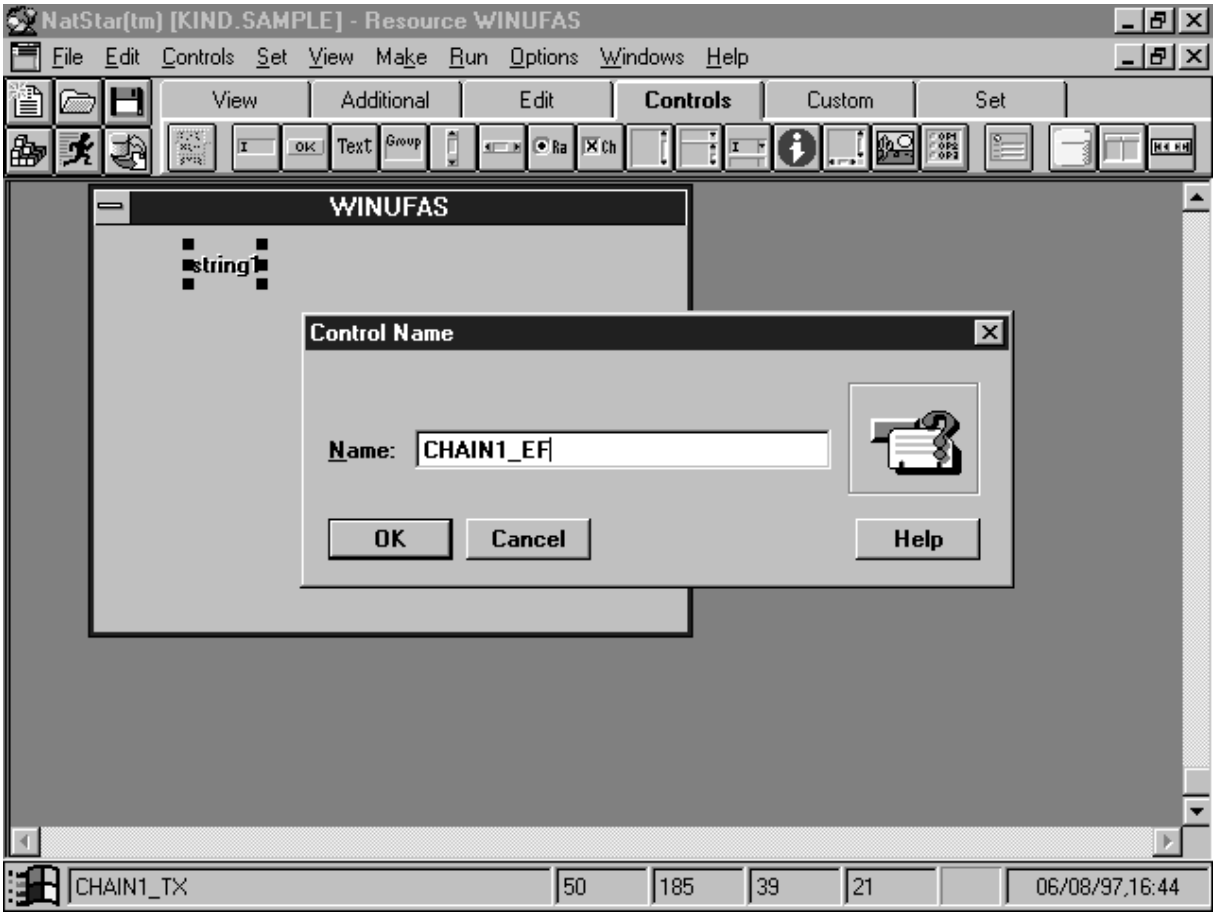

Figure C-6. Define the Entry Field for the First Operand

Click on **OK** to add the new control to the **WINUFAS** dialog window.

Double-click on the entry field **CHAIN1_EF**. The **Entry Field Info** window is displayed (see below).

Select default from the Type group

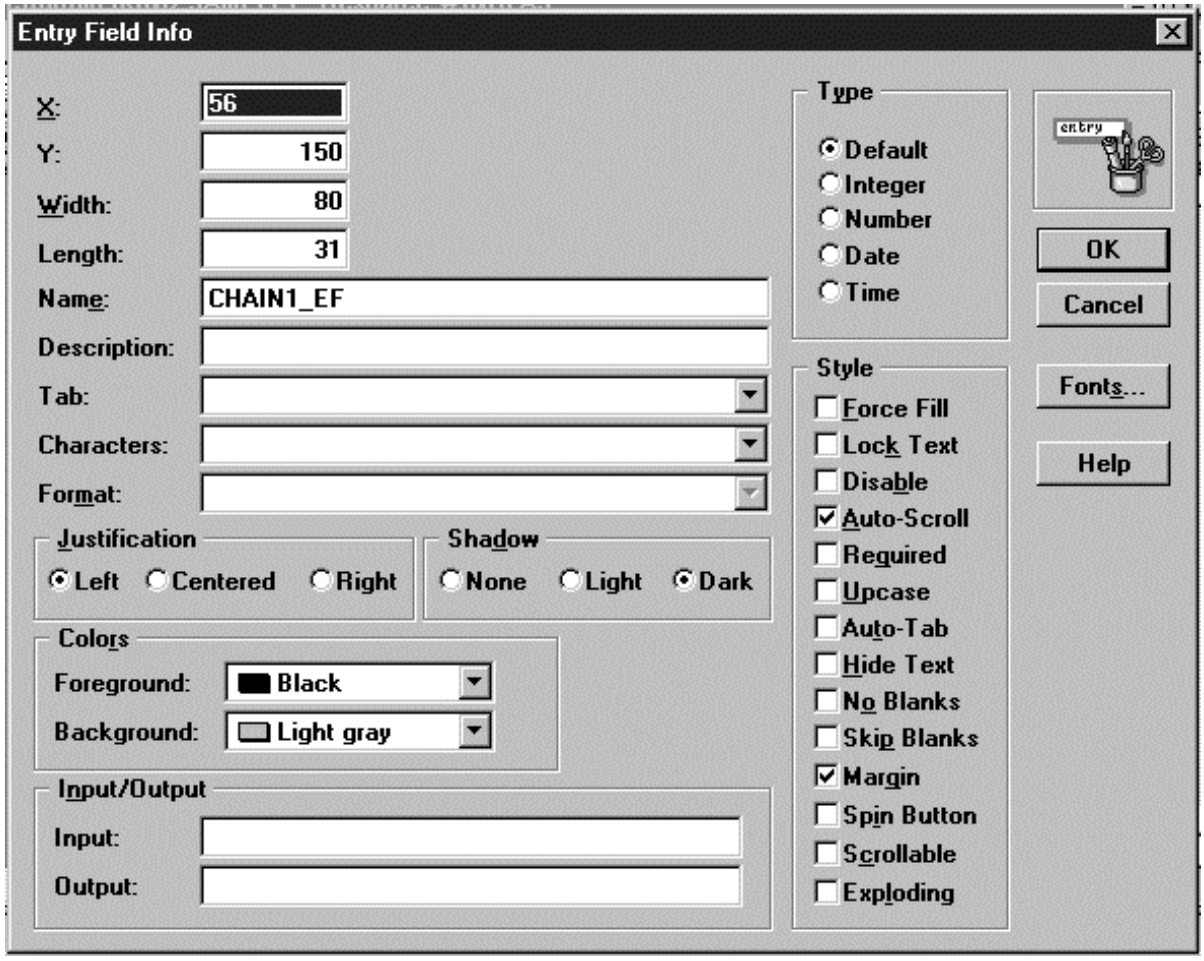

Figure C-7. Define the Entry Field Type for the First Operand

Click on **OK** to complete this definition.

Repeat the steps associated with Figures C-5, C-6 and C-7 to create the **CHAIN2_TX** and **RES_TX** text fields and the associated **CHAIN2_EF** and **RES_EF** entry fields. For the **RES_EF** field, when selecting its **Type** as **default** also select **Disable** from the **Style** group, so that the user cannot enter data in this result field.

Next, create the **CONCAT** push-button. To do so, select and drag the **Button** icon into the **WINUFAS** window. The **Control Name & Text** window is displayed (see below).

In the **Name:** field, enter **PB_CONC**. In the **Text:** field enter **CONCAT**.

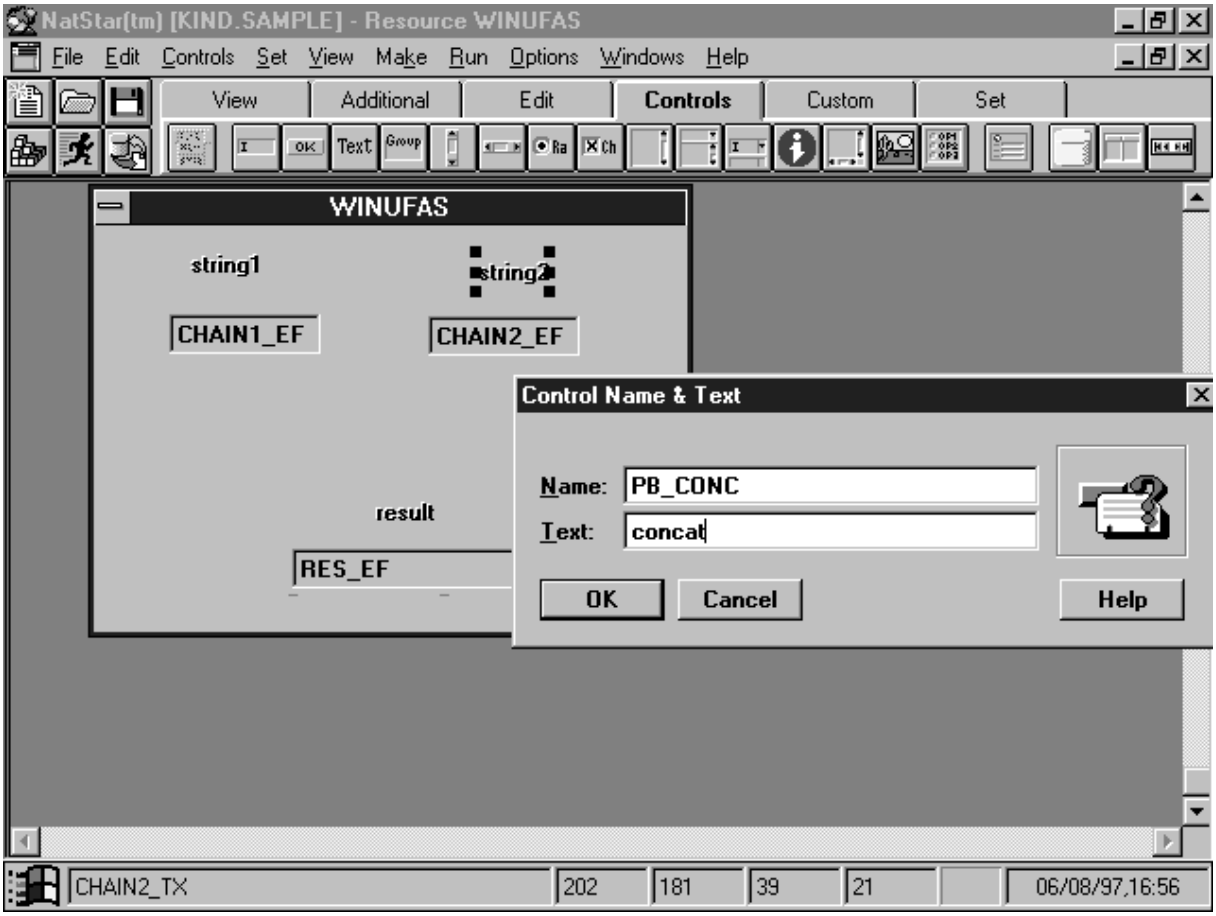

Figure C-8. Define the Concat Push-button

Click on **OK** to complete this definition.

You can use the alignment features of the NatStar Graphical Builder to position the text, entry fields, and push-buttons in a logical arrangement on the window. For information about how to use these features, refer to the section on Graphical Builder in the *NatStar User Manual -- Graphical Builder, Process Modeling, Information Modeling* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

When you have completed positioning the contents of the window, the **WINUFAS** window should resemble the one shown in the figure below.

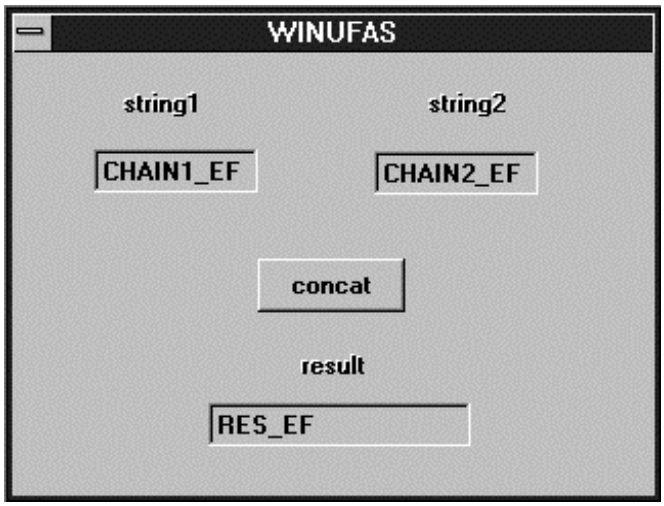

Figure C-9. Complete the User Interface Window

Save the **WINUFAS** window definition by selecting **Save** from the **File** menu or by clicking on the **SAVE** icon.

Close the window by selecting **Close** from the **File** menu.

C.3 Associating the Client and Server Libraries

The elements defined so far are related as follows. When the user activates the **Concat** button in the **WINUFAS** window, which is defined in the **TESTUFAS** library, the **SRV_CONC** function in the **SRVUFAS** library will be called. Because the **SRV_CONC** instruction is located in a different library than the **CONCAT** push-button, the library **TESTUFAS** must be configured to access the instruction **SRV_CONC** in the library **SRVUFAS**. If this relationship is not defined, the diagnostic **"invalid data name"** will be generated when the NCL for the **Concat** button is checked for correctness by using the **Check** icon.

To grant the **TESTUFAS** library access to the instructions in the **SRVUFAS** library, double-click on the **TESTUFAS** library in the **Libraries** structure. The **Modify Description for Library** window is displayed (see below).

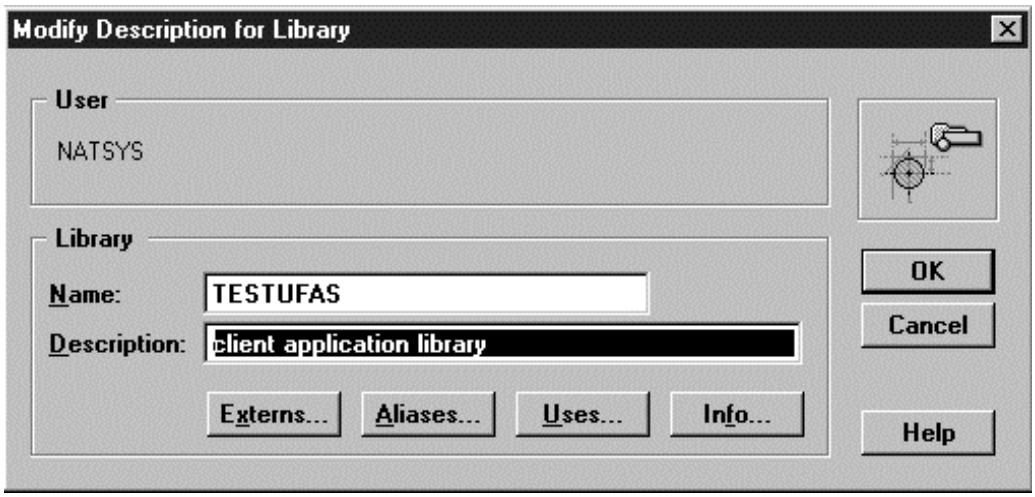

Figure C-10. Access the TESTUFAS Library Description

Click on the **Uses...** button.

The **Libraries used by TESTUFAS** window is displayed (see below).

Select the **SRVUFAS** library and click on **Select**. An asterisk (*) is displayed next to the library name.

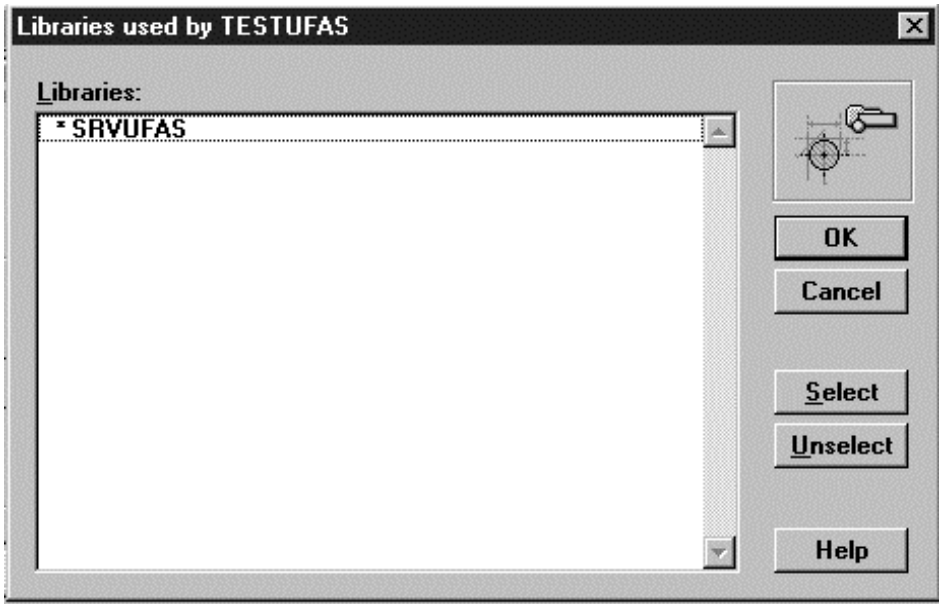

Figure C-11. Give TESTUFAS Library Access to SRVUFAS Library

Click on **OK**. When you are returned to the **Modify Description for Library** window, click on **OK** to complete this operation.

C.4 Entering NCL for the Client Logic

Next you need to enter the NCL to define the logic associated with Concat push-button.

Right mouse double-click on the **WINUFAS** window resource under the **TESTUFAS** library of the **Libraries** window (this window is shown in Figure C-4). When the **WINUFAS** window is displayed, right mouse click on the **Concat** push-button. The push-button selection list is displayed as shown below.

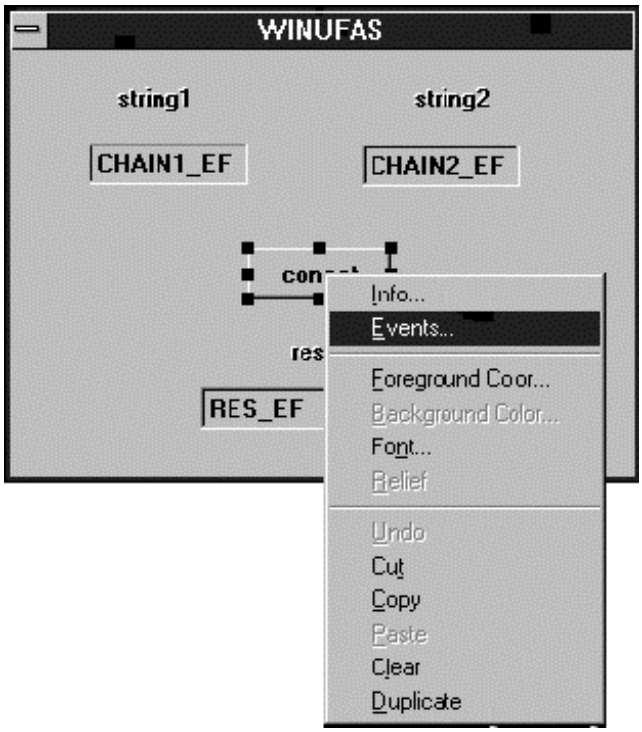

Figure C-12. Select the Executed Event for WINUFAS

Select **Events...** by clicking on that entry in the popup menu.

When the **Events for Resource WINUFAS.PB_CONC** window is displayed (see below), select the **Edit** tab. Select **EXECUTED** from the list of events. Enter the NCL for the **EXECUTED** event as shown in the figure.

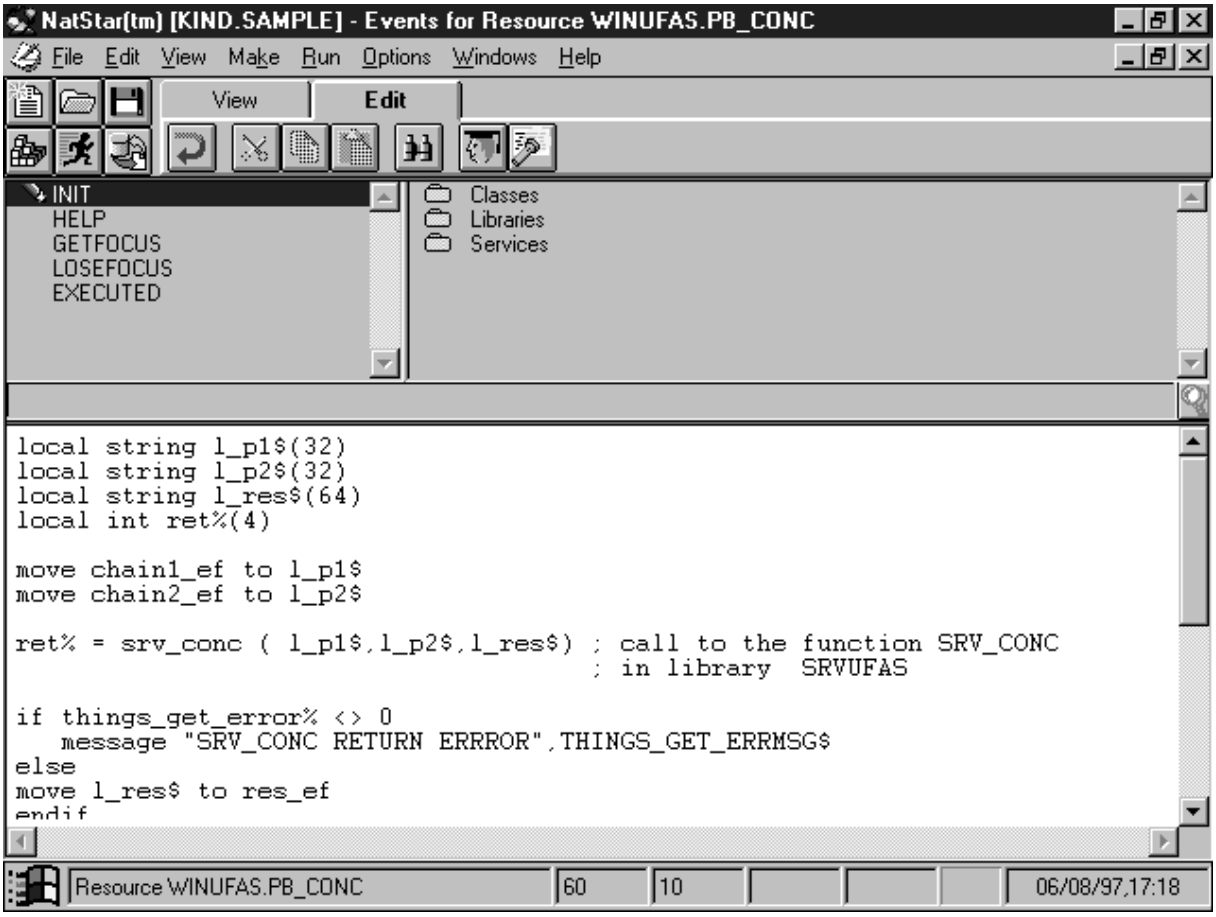

Figure C-13. Enter NCL for the Concat Push-button

The NCL you enter will be executed when the user clicks on the **Concat** push-button and the associated event is invoked.

NOTE:

This is a demonstration application, and so it does not include the diagnostic checks -- e.g., for valid input -- that are typically included in a production application.

Verify the correctness of the NCL by clicking on the **Check** icon. If the NCL is correct, **Resource WINUFAS.PB_CONC** is displayed in the status line.

Save the NCL by selecting **Save** from the **File** menu, or by clicking on the **Save** icon.

Close the **Events for Resource WINUFAS.PB_CONC** window by selecting **Close** from the **File** menu.

Checking for Errors After Call To Remote Instruction or Function

NatStar provided two functions that return a description of the current error:

```
THINGS_GET_ERROR%
THINGS_GET_ERRMSG$
```

You can call these functions after each call to a remote instruction or function, to determine if any error was detected -- not by the instruction or function, but by the NSCOM Communication Services internal routines.

The `THINGS_GET_ERROR%` function returns the error class, severity, error subclass, error code, and description of the error.

The `THINGS_GET_ERRMSG$` function returns the diagnostic for the error received by the `THINGS_GET_ERROR%` function. The diagnostic contains the cause of the error and a suggested solution.

Refer to *Handling Errors* in Chapter 5 for a discussion of how to handle errors of this type, including how to use these two functions. These functions are also discussed in the *Nat Systems documentation* (the order numbers of Nat Systems documents are provided in *Bibliography* in the *Preface* of this Guide).

Save the **WINUFAS** window by selecting **Save** from the **File** menu or by clicking on the **Save** icon. Close the **WINUFAS** window by selecting **Close** from the **File** menu.

Close the **Libraries** window by selecting **Close** from the **File** menu.

C.5 Defining the Configuration for the Client Window

Next you need to create a configuration for the **WINUFAS** window so that the application can execute on the client system. To do so, select the **Options/Configuration** menu item. The **Select Default Configuration** window is displayed (see below).

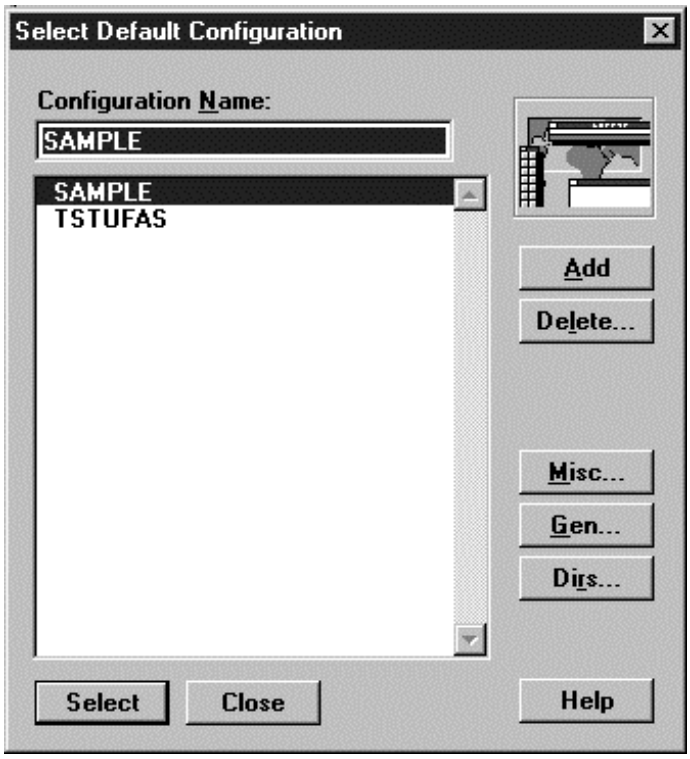

Figure C-14. Create the WINUFAS Window Configuration, Part 1

Enter **TSTUFAS** in the **Configuration Name** entry field, then click on **Add** to add **TSTUFAS** to the configuration.

With the **TSTUFAS** configuration still selected, click on the **Misc...** button. The **Miscellaneous** window is displayed (see below).

In the **Library** field enter **TESTUFAS**, and in the **Main Window** field enter **WINUFAS** and check **Client** in the **Kind** field.

Go to the **Targets** tab and verify that **LOCAL** is selected in all fields.

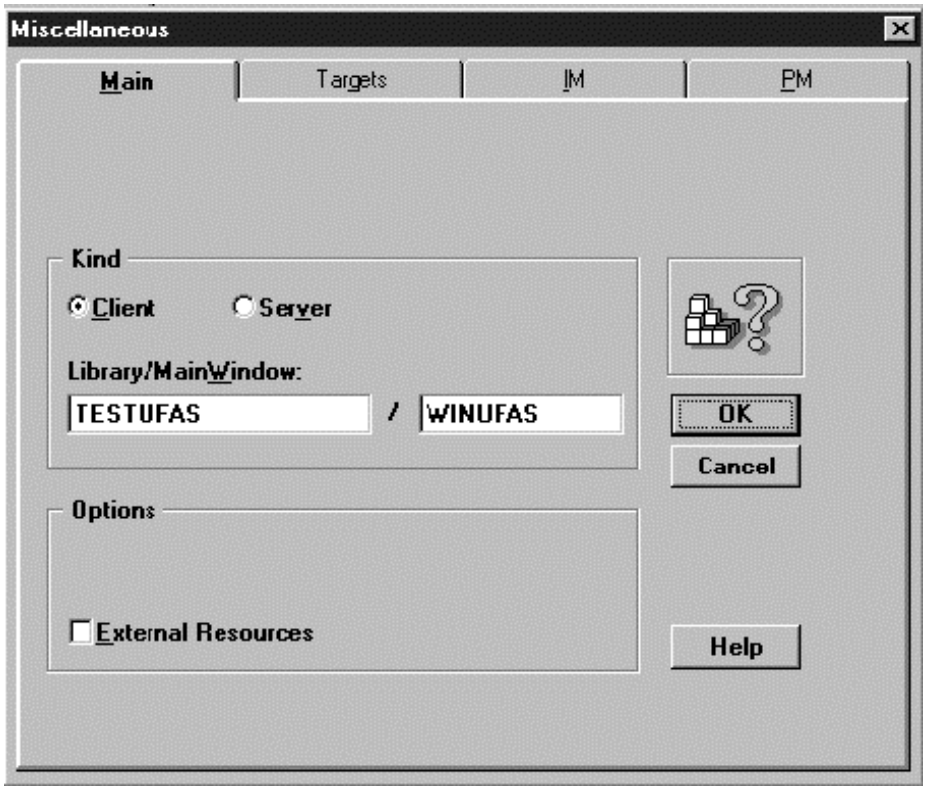

Figure C-15. Create the WINUFAS Window Configuration, Part 2

Click on **OK** to complete the configuration definition.

You are returned to the **Select Default Configuration** window (refer to Figure C-14). With the **TSTUFAS** configuration still highlighted, click on **Select** to assign **TSTUFAS** as the default configuration and close the **Select Default Configuration** window.

C.6 Building Resources for the Concatenate Application

The final step required before you are ready to test the **concatenate** application is to build the resources for the application. To do so, select **Build resources...** from the **Make** menu. The **Generator** window is displayed (see below).

Select the **L.NATSYS**, **L.TESTUFAS**, and **L.SRVUFAS** libraries in the left box, and click on **Add>>** to move them to the right box, as shown in the figure.

Verify that the **Setup** box ***is*** checked, and that the **Generate DLL** and **Debug** boxes are ***not*** checked.

Click on **Build** to build the resources.

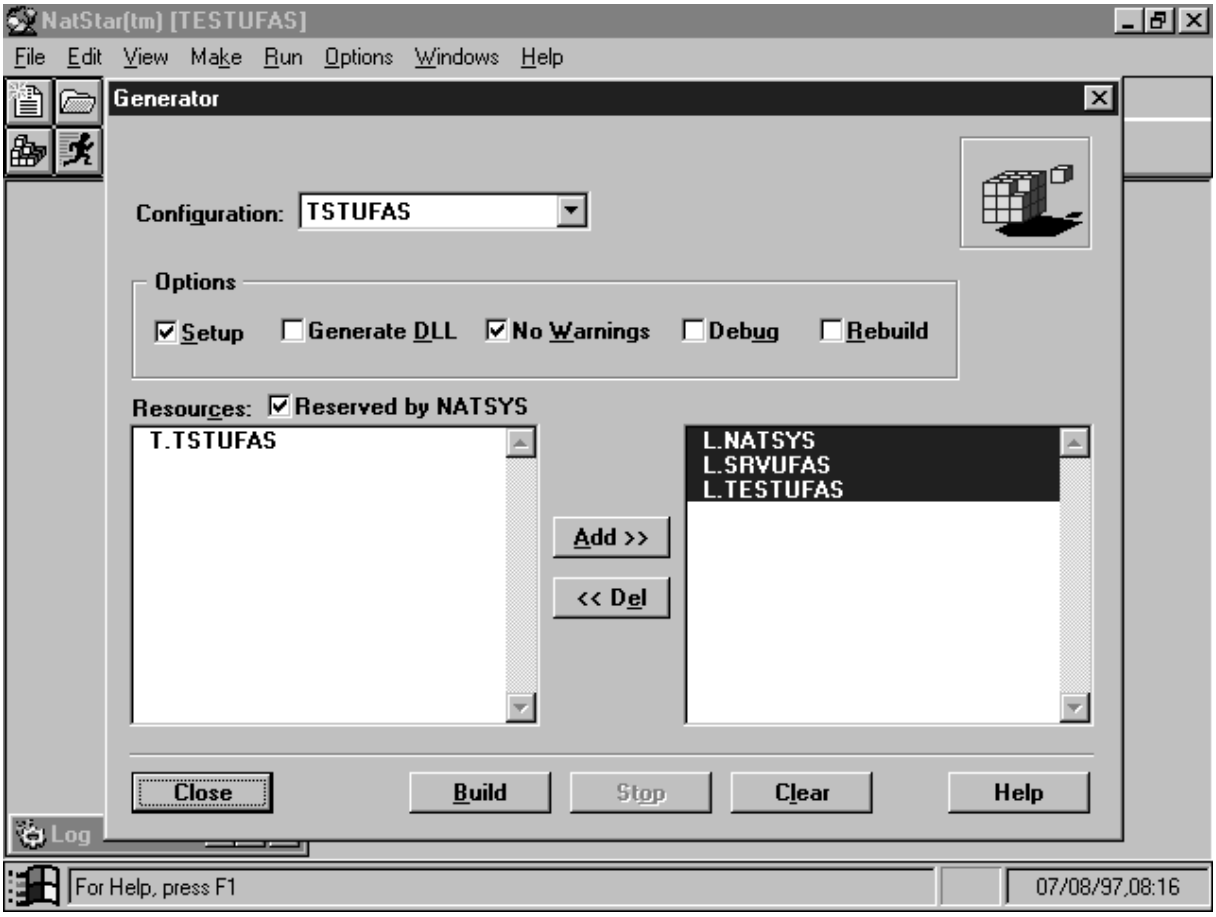

Figure C-16. Build Resources for the Concatenate Application

Click on **Close** to close the **Generator** window.

After you close the **Generator** window, check the **Log** file to verify that no errors occurred while building the resources. Do so by selecting the **Log** entry from the **Windows** menu, or by clicking on the **Log** icon.

C.7 Testing the Application

To test the **concatenate** application, select **Start** from the **Run** menu or click on the **Run** icon. The **WINUFAS** window will be presented (see below).

You can enter operands and observe the results. At this point, the entire client/server application is being executed in interpretive mode using the NCL entered as explained above.

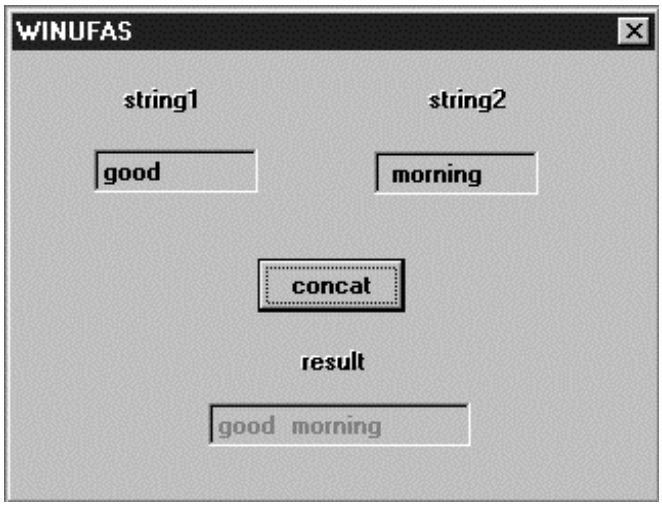

Figure C-17. User Interface for Concatenate Application

D. GCOS 7 Configurations

After you have tested the client and server logic on the NatStar development workstation, you can define the GCOS 7 configuration. The basic configuration will identify GCOS 7 as a User Interface target for this example.

NOTE:
Some of the screens reproduced here show only part of a file. For the full text, see Appendix G *NCL Sequence Listings*.

D.1 Configuring GCOS 7 as a NatStar Server

The first step is to configure GCOS 7 as a NatStar server system.

To do this, you need to define 2 targets for GCOS 7:

- Language, and
- User Interface.

Select the menu **Targets** from the **Options** menu.

The following figure shows the screen after you have completed this step.

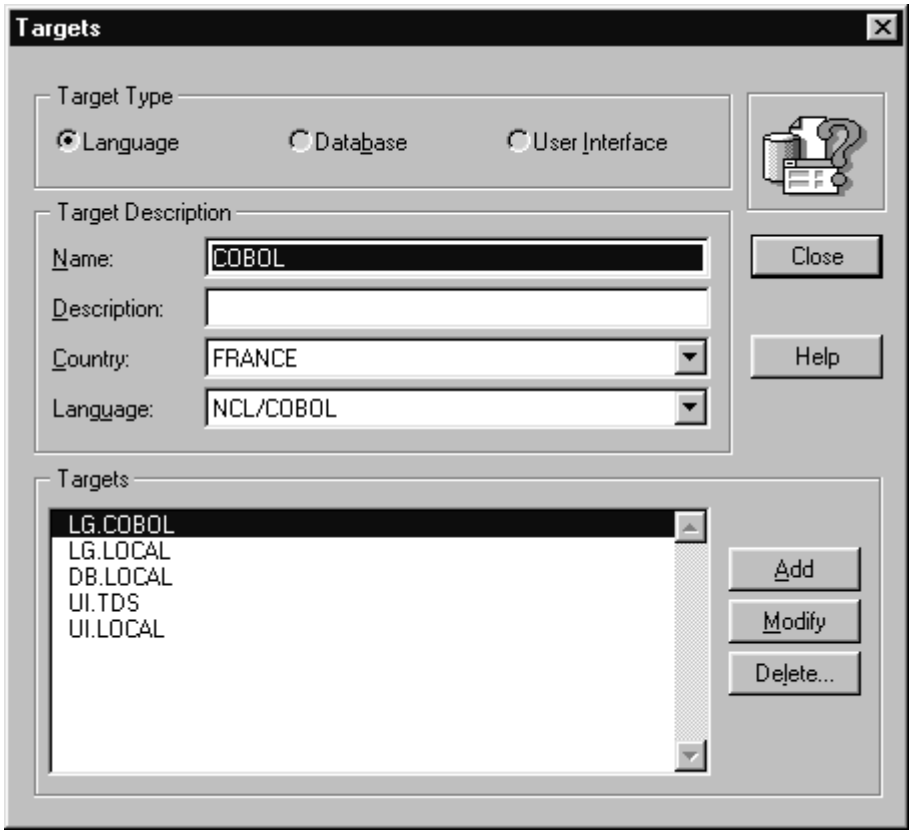

Figure D-1. Creating the Language Target Type

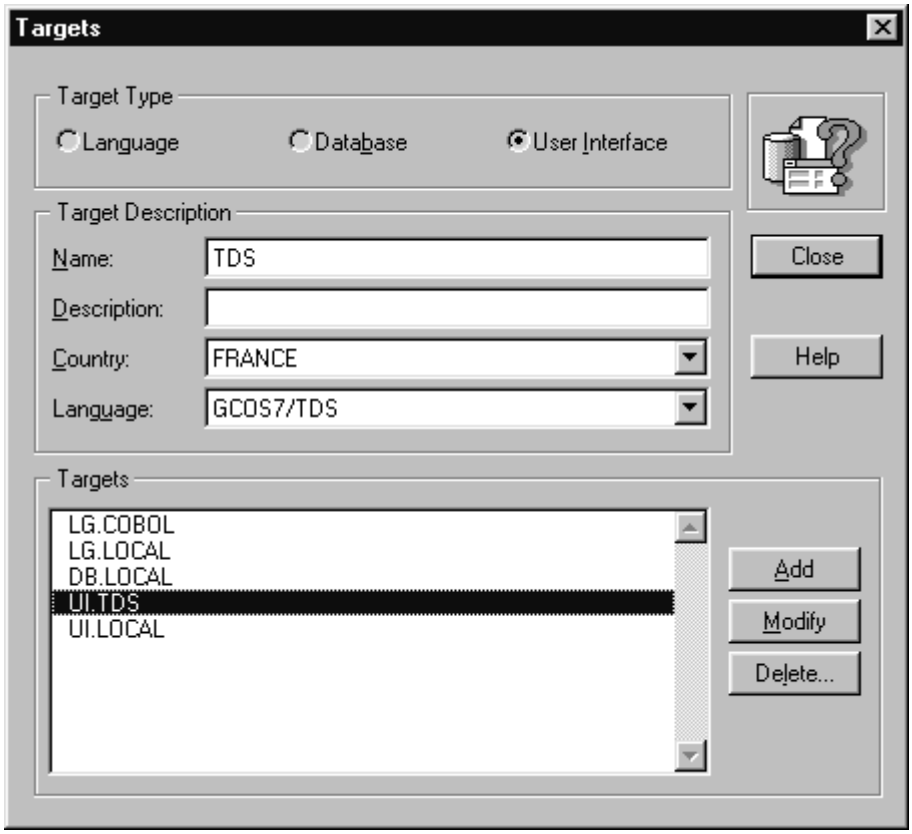

Figure D-2. Creating the User Interface Target Type

Click on **Close** to leave the **Targets** definition window.

D.2 Defining the Configuration for COBOL-85 Generation

Select the **Options/Configuration** menu, which causes the **Select Default Configuration** window to be displayed (see below).

In the **Configuration Name:** field, enter **SRVUFAS**. Click on the **Add** button to define the new configuration.

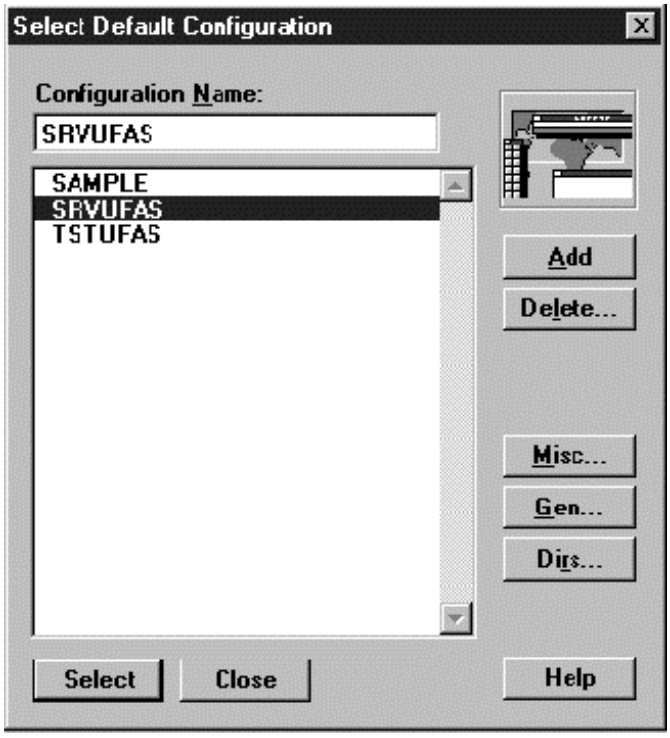

Figure D-3. Define Configuration for COBOL-85 Generation

With the SRVUFAS configuration highlighted, click on the **Misc...** button.

The **Miscellaneous** window is displayed (see below).

Select the **Main** tab, then select **Server** in the **Kind** group.

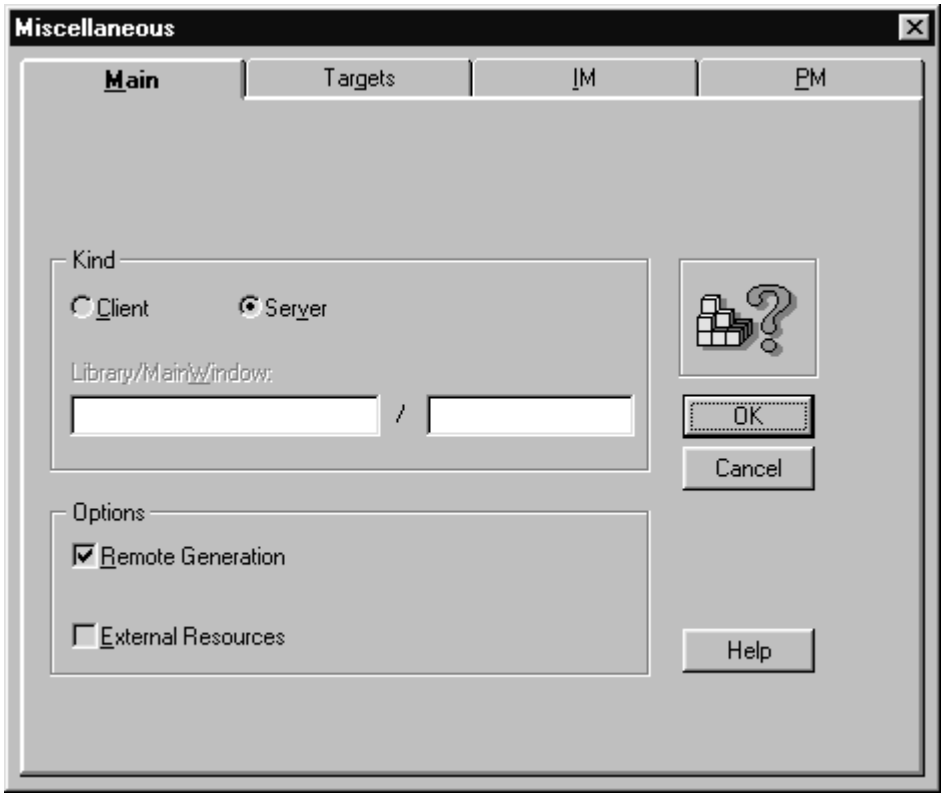

Figure D-4. Define COBOL-85 Generation as a Server Configuration

Choose the **Targets** tab to define the user interface to be used in this configuration (see below).

In the **UI Target** field, select **GCOS 7**, which you defined earlier (refer to Figure D-2).

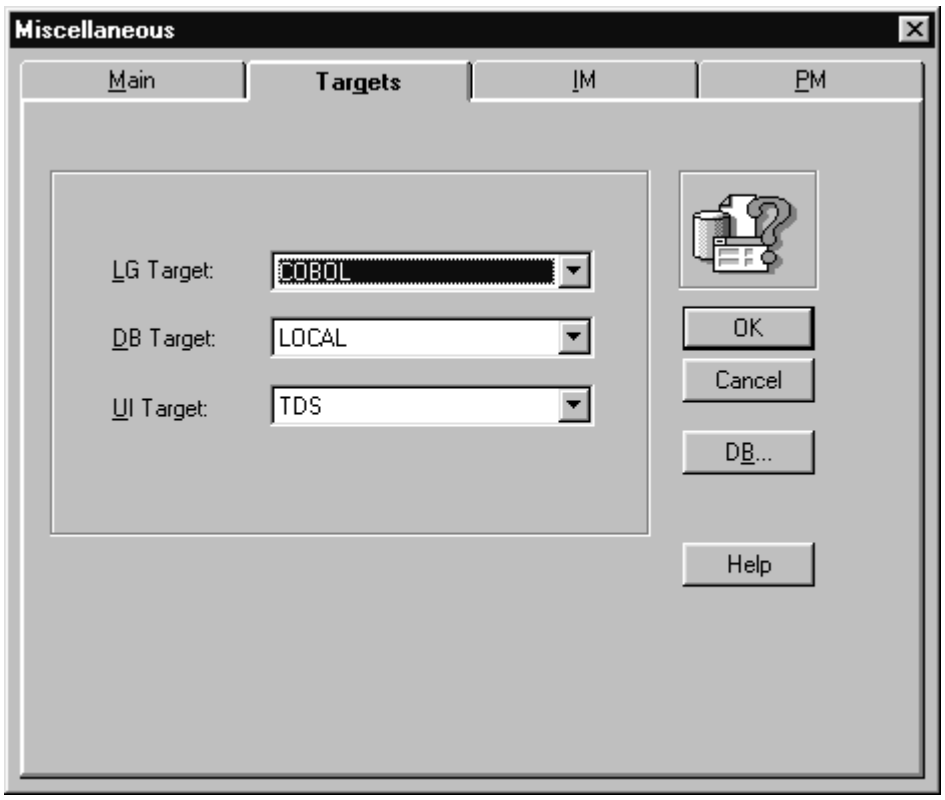

Figure D-5. Link COBOL-85 Generation Configuration to the User Interface Definition

Click on **OK** to save this configuration definition.

Return to the **Select Default Configuration** window.

With the **GCOS 7** configuration still highlighted, click on the **Dirs...** button, which causes the **Directories** window to be displayed (see below).

Enter the pathnames on the PC where you want the generated COBOL-85 source to be stored, using the example names shown in the figure. The COBOL-85 source will be stored in the pathname defined in the **Source:** field, and the COPY source will be stored in the pathname entered in the **Include:** field.

The values in the directories **Objects:**, **Binaries:**, **DLLs:**, and **Libraries:** are not used during COBOL for GCOS 7. The values in the figure are provided only for completeness. All of the directories are used during C generation if executables (.EXE and .DLL files) are to be built for execution on the client system.

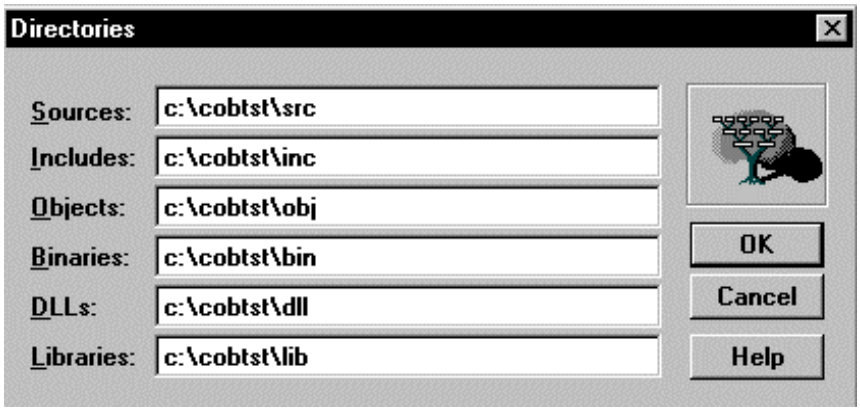

Figure D-6. Define Pathnames for COBOL-85 Source

Click on **OK** to define the new configuration. You are returned to the **Select Default Configuration** window.

Click on the **Close** button to accept the values for the configuration and close the **Select Default Configuration** window. You must click on **Close** instead of **Select** because a configuration defined as a **Server** configuration cannot be selected as the default configuration (which is implied by the **Select** button).

D.3 Introducing the Connect Disconnect Phase for Remote Servers

First, import the library G7USRPASS from the cdsonn.exp file. This is located on the OPEN 7 partition of the delivery in `/usr/natsys/appli/cdsonn.exp`. You must transfer this file onto your PC before importing the library, with the binary option of FTP.

Click on the import icon of the menu bar, and give the location of the import file.

Perform the import with the automatic option of the IMPORT command.

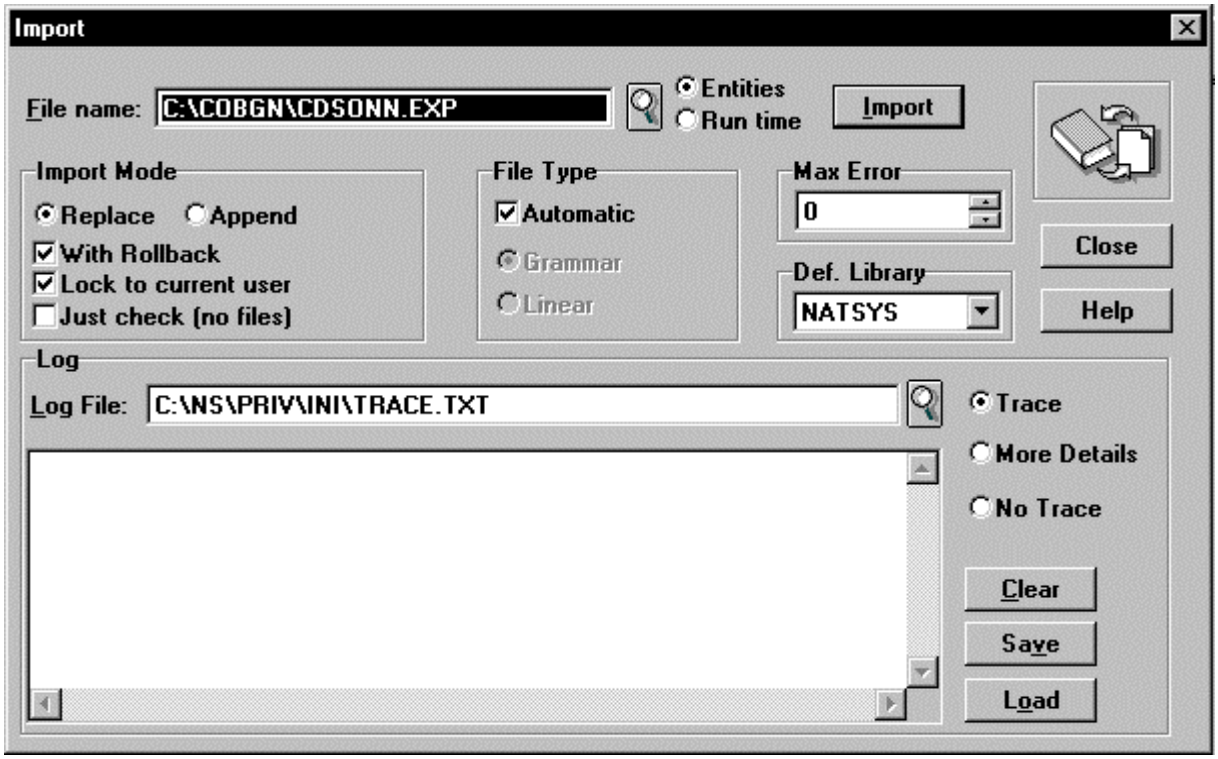

Figure D-7. Importing the G7USRPASS library

When the command is complete, you should have the new G7USRPASS library in the list of libraries.

To insert the templates for CONNECTION and DISCONNECTION in the WINUFAS main window:

1. Associate the G7USRPASS library with TESTUFAS by using the USES push button of the Modify Description for Library window (refer to Figure C-10), so that TESTUFAS can access the template inside the G7USRPASS library.
2. For each template, G7CONECT and G7DISCNT, included in the G7USRPASS library, manually modify the name of the symbolic server to which you wish to connect.

   In the event EXECUTED for the two push-buttons for CONNECT and DISCONNECT change the string for the symbolic server name to the name you require (give the same name as in Section E.1 *Defining the Remote Servers*), in the instruction THINGS_SET_DEFAULT_SERVER.
3. To edit the WINUFAS line in the libraries list, double click on it with the right mouse button.

You may reorganize the entry field already inserted for the CONCAT operation.

Click on the CONTROLS tab, select and drag the TEMPLATE icon to the WINUFAS window.

Give TP_CONN as the name of the template and G7CONECT for the template itself.

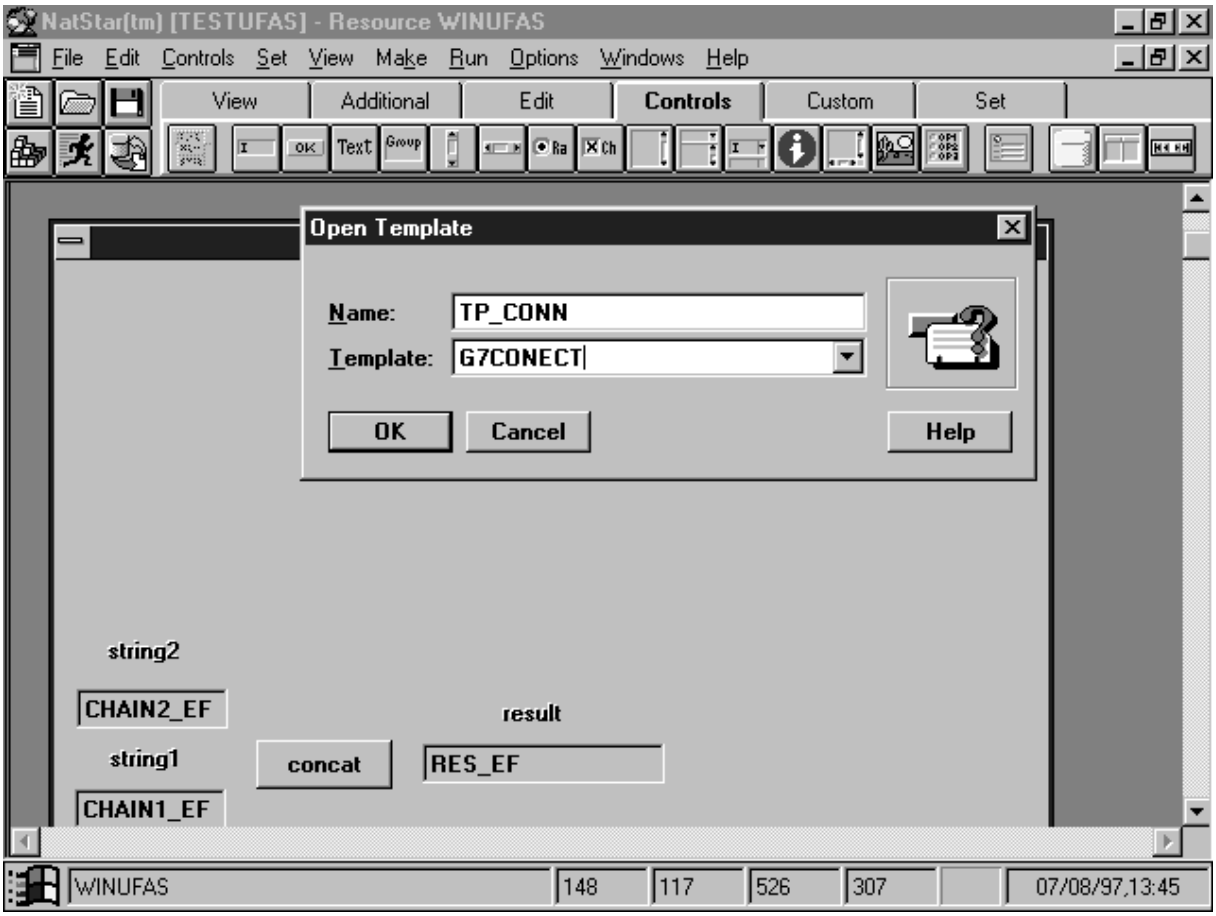

Figure D-8. Defining the Connection Template

The appearance of the template introduced is only a frame, which can be moved wherever you want (the entry fields of the template are visible at execution time).
Repeat the same operation with the second template:
name = TP_DISCNT
template = G7DISCNT

Refer to Figure D-8 above.

D.4 Introducing the UFAS Access File

This part cannot be tested in local execution because UFAS access is not available on PCs.

In order to activate the CHECK without error, copy a file describing the UFAS API into the working directory of NatStar.

In the OPEN 7 partition delivery `/usr/natsys/`, transfer the `/usr/natsys/appli/ns02ufas.ncl` onto the PC, in the `($NATSTAR)\GLOB\SERVICES\NCL` directory.

In this example, there is only a read access to a UFAS file, so it is necessary to modify the WINUFAS application window to introduce one entry field for the key file, the result entry field, and an action push button. The UFAS file is controlled by TDS, and the EFN and IFN are assigned together in the STDS file given as an example in SI7.NS7.SLLIB.

With the WINUFAS window displayed on the screen, drag successively from the CONTROLS tab:

- Entry field USER_EF with default characteristics (this will be the return of the UFAS access function) and put it in disable mode.
- The text COUNT_TXT with the string "No COUNT".
- The entry field NO_COMPT_EF with default characteristics.

Push button PB_LISTE with « read file » string label, and the following code in its executed event.

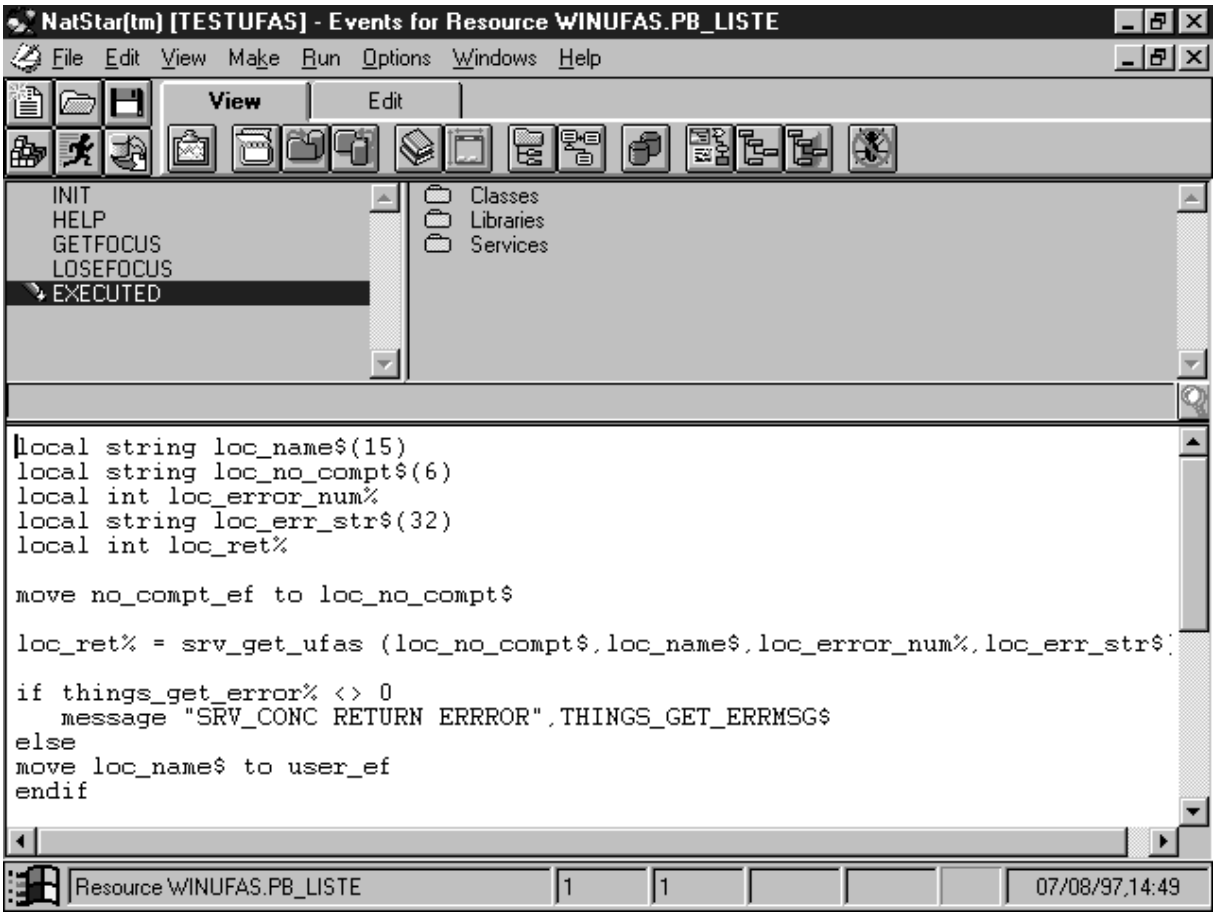

Figure D-9. NCL Code in EXECUTED Event of PB_LISTE Push Button

D.5 Creating a New Server Remote Function

Create the SRV_GET_UFAS remote function, in the SRVUFAS library. Refer to Figure B-4 to insert a new resource.

1. Name of the resource SRV_GET_UFAS.
2. Description = access UFAS function.
3. Check the FUNCTION from KIND group.
4. Click on OK to complete the definition of the new SRV_GET_UFAS function.

Now define the parameters function (double click on the SRV_GET_UFAS line from the list libraries), select the PARAMETERS tab, and enter the following description.

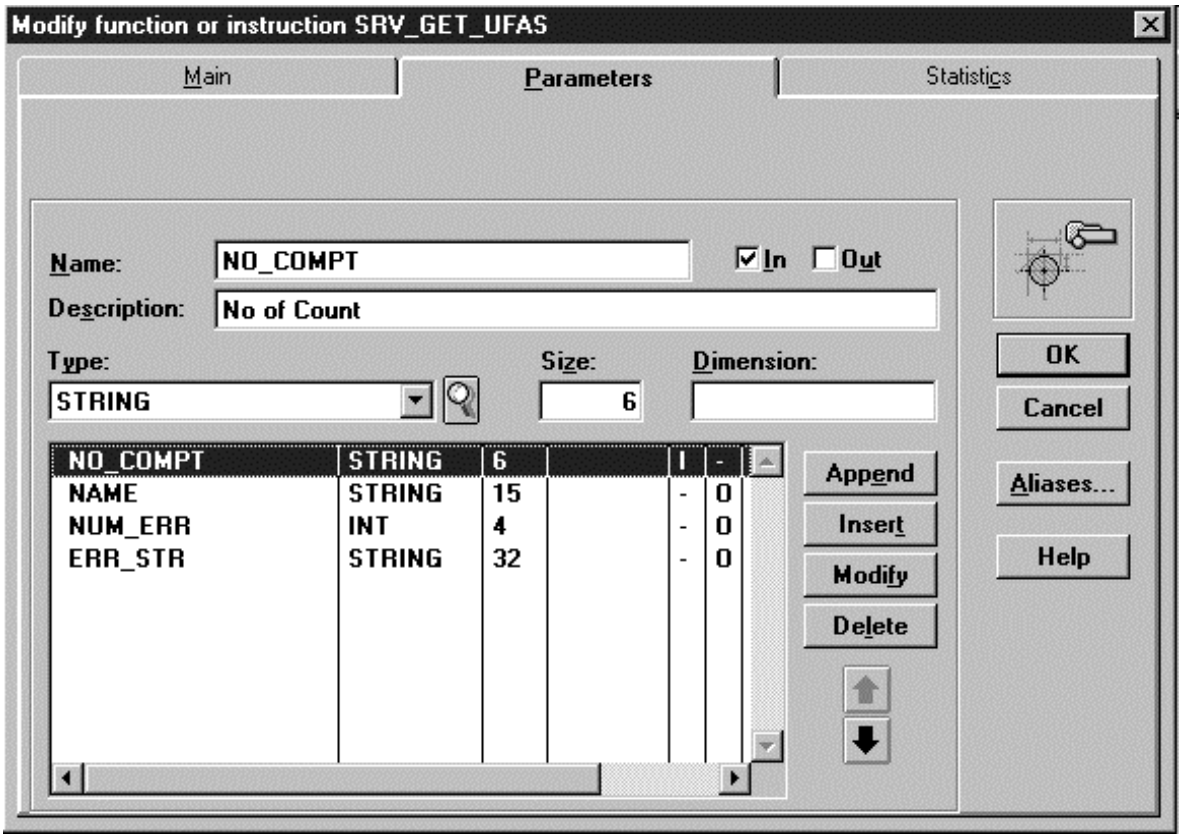

Figure D-10. SRV_GET_UFAS Function, Parameters Definition

After you have entered all of the parameters, click on OK to complete the definition of parameters.

You are now ready to enter NCL describing the access to UFAS file on GCOS 7. Double click right on the SRV_GET_UFAS function, under the library SRVUFAS.

The EDIT window of the SRV_GET_UFAS function is displayed, as shown below. Enter the NCL that will be executed on GCOS 7, when the function will be called.

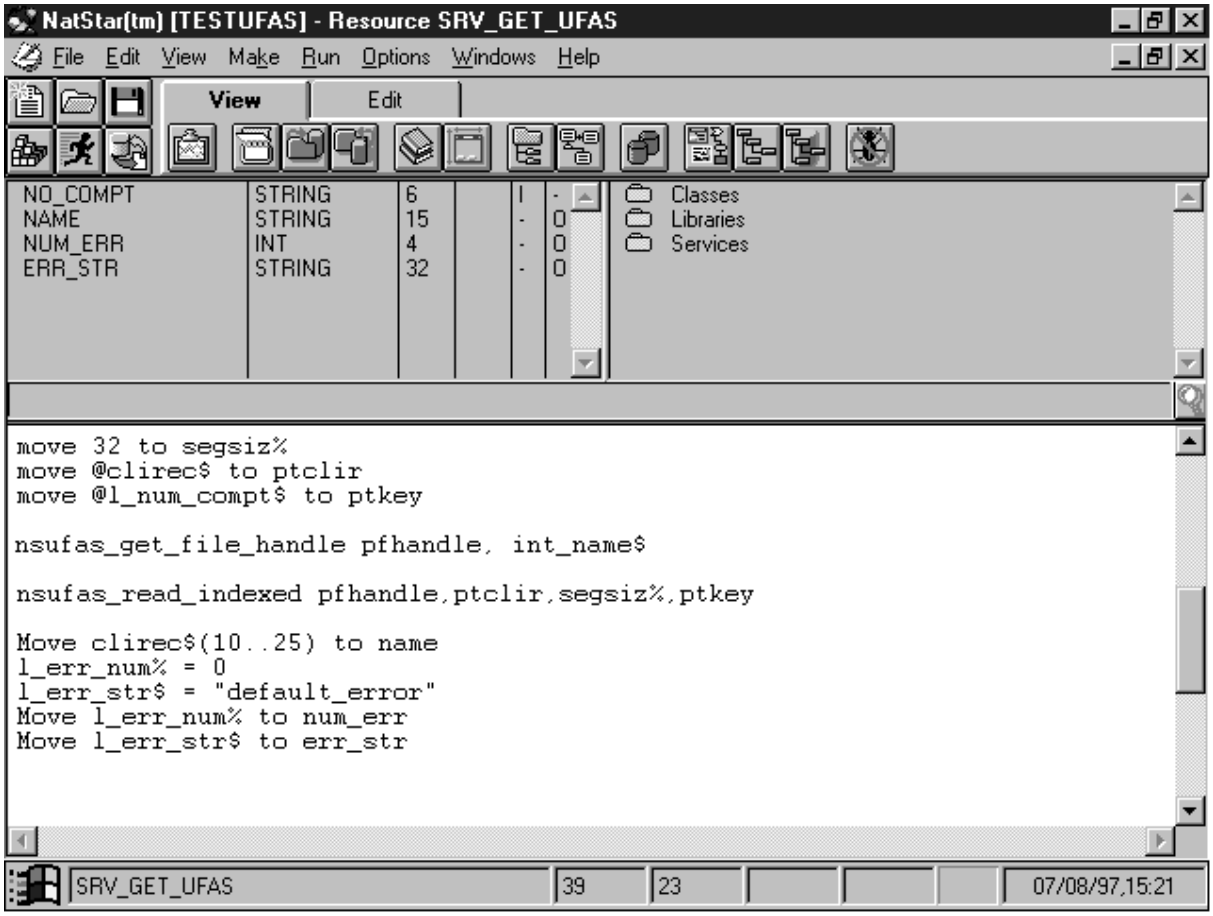

Figure D-11. Displaying the Code Executed on the Server Side for the UFAS Access

All the code is not displayed on this screen (but can be found in the example.exp file under SRV_GET_UFAS function definition. This file delivered as an example in the OPEN 7 partition `/usr/natsys/appli/example.exp` contains the full example that being creating here).

Verify that the NCL is correct by clicking on the Check icon. This icon activates a check of the syntax of the NCL. If necessary, refer to the Nat Systems documentation for an explanation of errors that occur during syntax checking.

Any errors that occur during syntax checking appear in the status line of the display. If the NCL is correct, the instruction name, SRV_GET_UFAS, appears in the status line, as shown in the figure.

Save the NCL by selecting Save from the File menu, or by clicking on the Save icon.

Close the resource SRV_GET_UFAS by selecting Close from the File menu.

E. Partitioning the Application

The next step is to partition the logic so that the functions **SRV_CONC**, **SRV_GET_UFAS** execute on GCOS 7 as COBOL-85 TPRs. The window and associated logic will execute on the PC as the client part of the application.

E.1 Defining the Remote Servers

The first stage is to define the remote server that the client application can access. You do so by using THINGS_SET_DEFAULT_SERVER. This instruction specifies the logical name of a server, and is entered in the **INIT** event of the **Events for Resource WINUFAS** window.

In the **Libraries** window, right-mouse double-click on the **WINUFAS** icon (this is shown in Figure C-4). When the **WINUFAS** window is displayed, right-mouse click on the title bar. The **Events for Resource WINUFAS** window is displayed. Select the **INIT** event (see below). Enter the NCL shown in the figure.

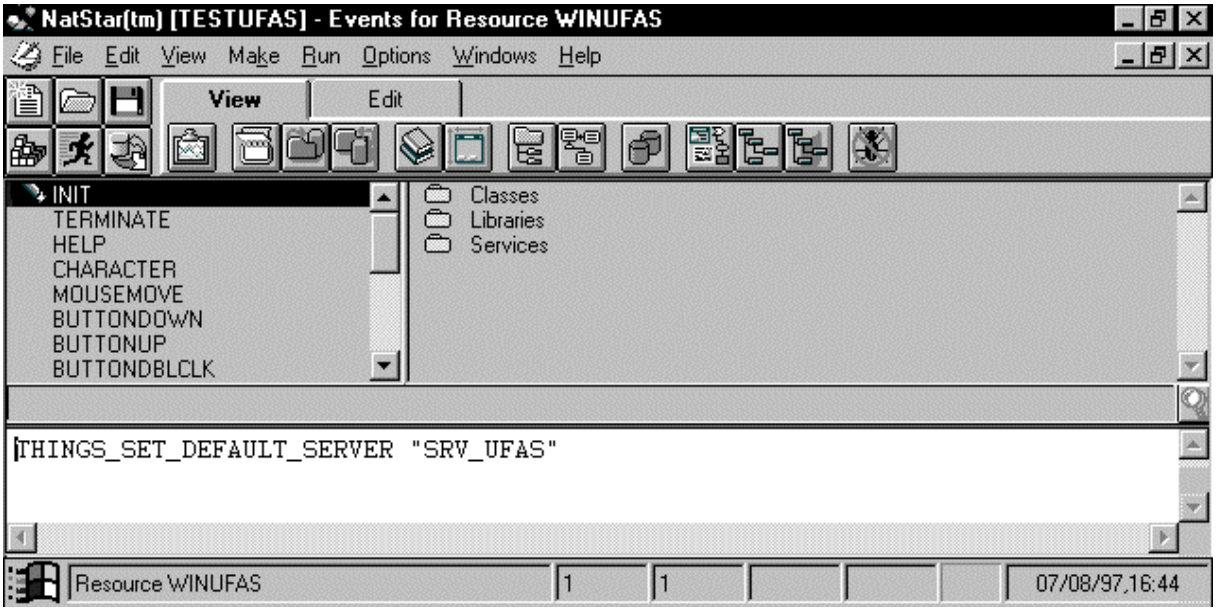

Figure E-1. Enter NCL for WINUFAS INIT Event

Verify the correctness of the NCL you entered by clicking on the **Check** icon. If the NCL is correct, save the NCL by selecting **Save** from the **File** menu or by clicking on the **Save** icon. Close the **Events for Resource WINUFAS** window by selecting **Close** from the **File** menu.

Save the **WINUFAS** window by selecting **Save** from the **File** menu or by clicking on the **Save** icon. Close the **WINUFAS** window by selecting **Close** from the **File** menu.

Close the **Libraries** window by selecting **Close** from the **File** menu.

E.2 Specifying Remote Operations

Next you need to specify how remote operations will be handled. First retrieve the configuration for the **concatenate and UFAS** application by selecting **Configuration** from the **Options** menu.

Select the **TSTUFAS** configuration and click on the **Misc...** button. The **Miscellaneous** window is displayed.

Select the **IM** tab and enter **NS02COMT** for the **Transaction Monitor** (see below).

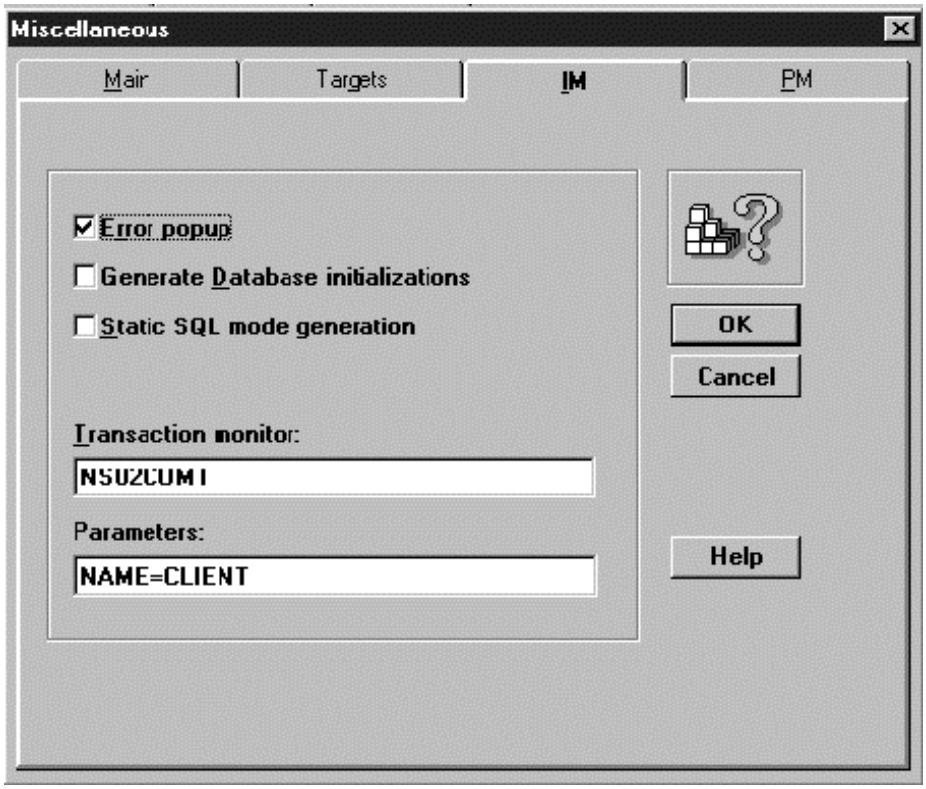

Figure E-2. Specify Remote Operations for the WINUFAS Calculate Application

Check the **Error popup** box if you want to have any diagnostics from the NatStar Communication Services software displayed in a diagnostic window when the client/server application is executing.

Normally you would use Error popup while debugging the client/server application, then turn it off for production operation (refer to *Handling Errors* in Chapter 5).

Click on **OK** to accept the defined configuration. You are returned to the **Select Default Configuration** window. With the **TSTUFAS** configuration still highlighted, click on **Select** to accept the values entered and close the **Select Default Configuration** window.

Note that in this case you must close the configuration by clicking on **Select**, which identifies **TSTUFAS** as the default configuration.

E.3 Identifying Instructions as Remote

You need to identify the instructions that will be executed remotely. In this case, they are the instructions **SRV_CONC**, **SRV_GET_UFAS**. To do so, right mouse click on the + icon associated with the **SRVUFAS** library to expand the list of instructions and functions associated with **SRVUFAS**.

Double click with the Left mouse button on each instruction to display the **Modify function or instruction** window (see below).

In each window, check the **Remote** box of the **Options:** entry as shown in the figure. The **Remote** box indicates that the function or instruction will be executed remotely as the result of a call from the client application, and determines how the source code in COBOL-85 is generated.

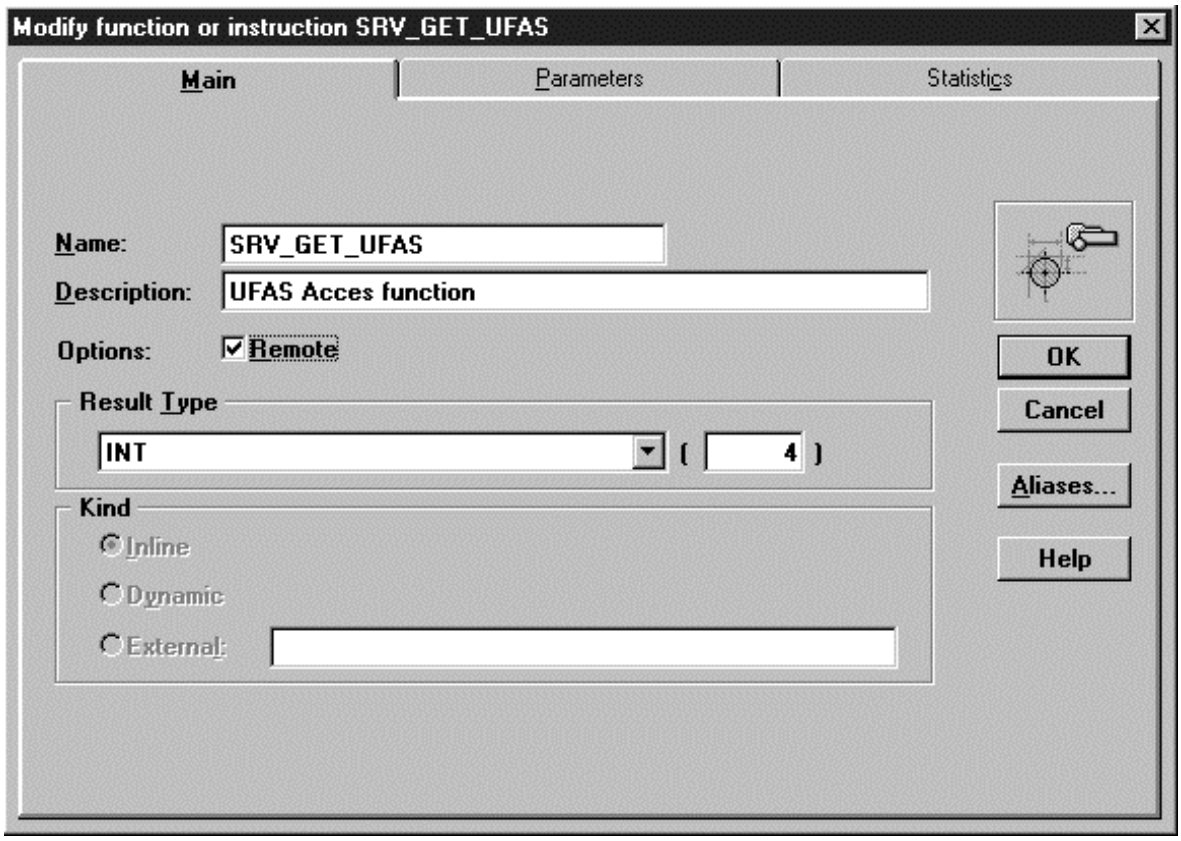

Figure E-3. Define Instruction as Remote

Click on **OK** to close the window.

NOTE:

Refer to the discussion of Remote and Local procedures Section 5.4 *Generating Server Source Code* for a more complete explanation of when to define an instruction or function as Remote.

The NCL code for **only** the Remote procedures that will execute on GCOS 7 must be generated (in COBOL-85), then transferred to the server where they are compiled and linked. Once this has been done, the remote procedures can be called from a NatStar client application.

You do not need to generate code for the client application at this time. It can remain in NCL form to be executed and debugged in interpretive mode. Typically you will leave the client application in NCL form until you are ready to distribute copies for production operation. At that time you can generate code and create executables to be distributed to all client systems that will use the client/server application.

Later in this example the **SRVUFAS** library for the GCOS 7 server will be rebuilt and the source code will be generated.

First, however, it is necessary to rebuild the portion of the concatenate, and srv_get_ufas application that will execute on the client.

Select **Build resources...** from the **Make** menu. The **Generator** window is displayed (see Figure E-4).

Select **TSTUFAS** from the list of configurations in the **Configuration:** entry as shown. Use the **Add>>** and/or **<<Del** buttons as necessary to make **L.TESTUFAS, L.G7USRPASS** and **L.NATSYS** the only entries in the right side of the **Resources:** list boxes.

Click on **Build** to rebuild the client configuration.

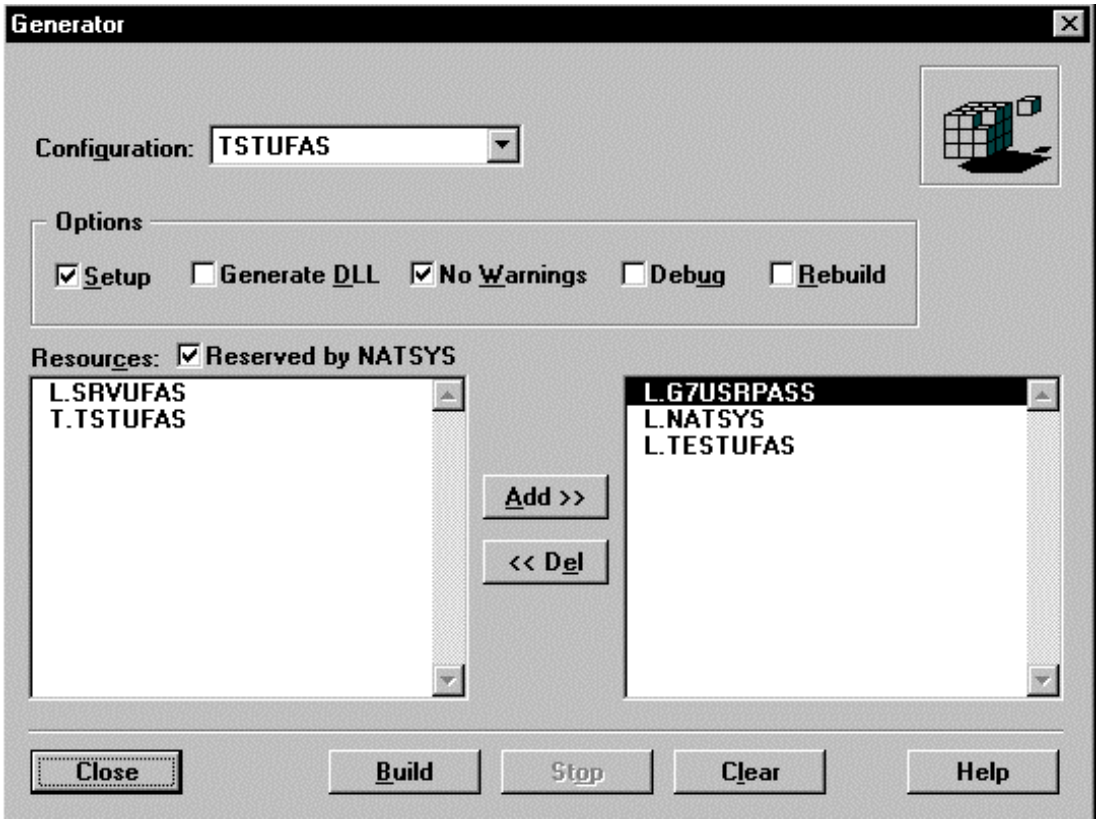

Figure E-4. Rebuild the Client Configuration

Close the **Generator** window by clicking on **Close**.

After you close the **Generator** window, check the **Log** file to verify that no errors occurred while building the resources. Do so by selecting the **Log** entry from the **Windows** menu, or by opening the **Log** icon.

F. Generating COBOL-85 Source Code

The next step is to generate the source code that will become the NSCOM7 TPR. You will generate code in COBOL-85.

F.1 Generating COBOL-85 Source

The NCL generator adds RPC stubs and other housekeeping code to the NCL code you wrote for this example, generates the source code in COBOL-85, and transfers it to GCOS 7.

Select **Build resource...** from the **Make** menu, or click on the **Build** icon. The **Generator** window is displayed (see below). Select the library that includes the remote procedures -- **SRVUFAS** in this example -- and move it to the right side of the screen by clicking on **Add>>.**

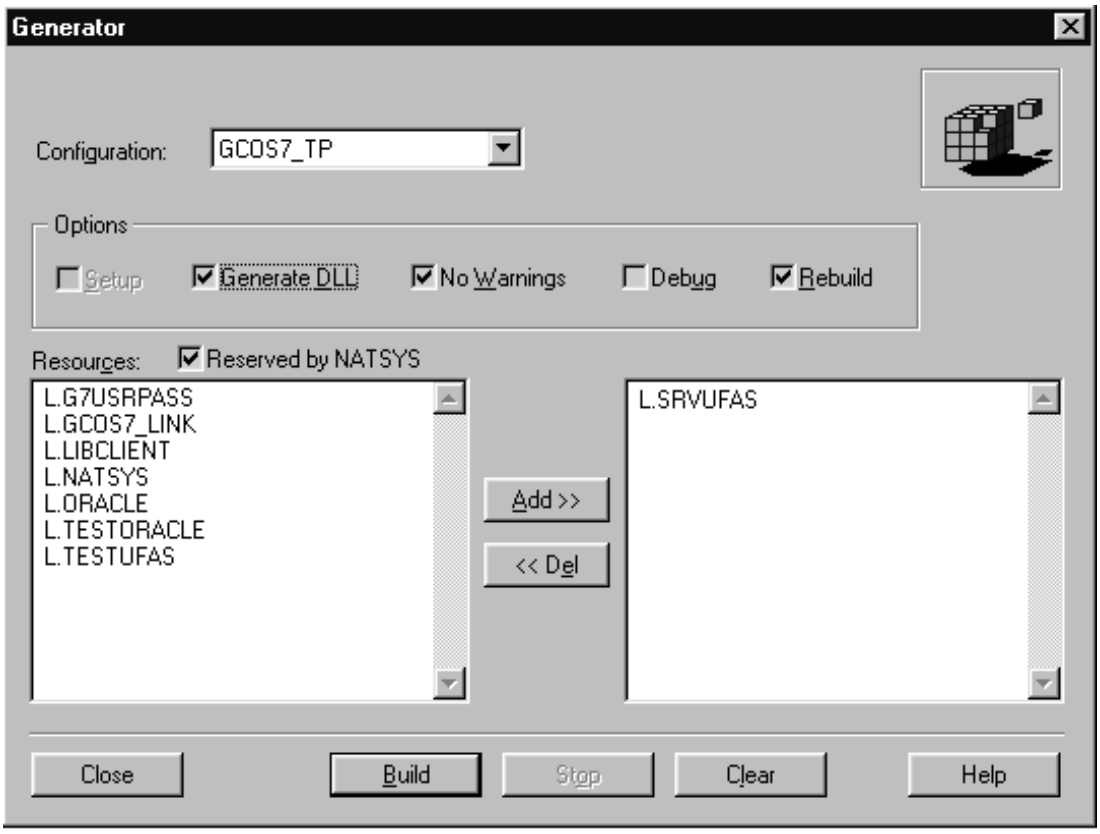

Figure F-1. Select SRVUFAS Library for NCL Generation

Make sure that the **Debug** box is ***not*** checked. Usually the **Debug** box is dimmed and cannot be checked, but be sure to verify its status.

Select **SRVUFAS** from the **Configuration** scroll box, and click on **Build** to invoke the NatStar NCL generator. When generation is complete, the text in the Status line at the bottom of the NatStar window stops changing. Close the **Generator** dialog box by clicking on **Close**.

Select the **Log** entry from the **Windows** menu, or click on the **Log** icon, to check the Log file to verify that no errors occurred. If no errors are listed in the Log window, exit from NatStar.

F.2 Preparing the Transaction Generation on GCOS 7

F.2.1 Compile Time

When all the files are ready, use the NSCOB7 job to compile the application.

Below is an example of how to run this job. All the parameters are documented in the NSCOB7 source itself.

The name of the first TPR of the transaction created by NATSTAR must be specified in the following windows.

Modifying Library Description

Select a library, then do EDIT/MODIFY:

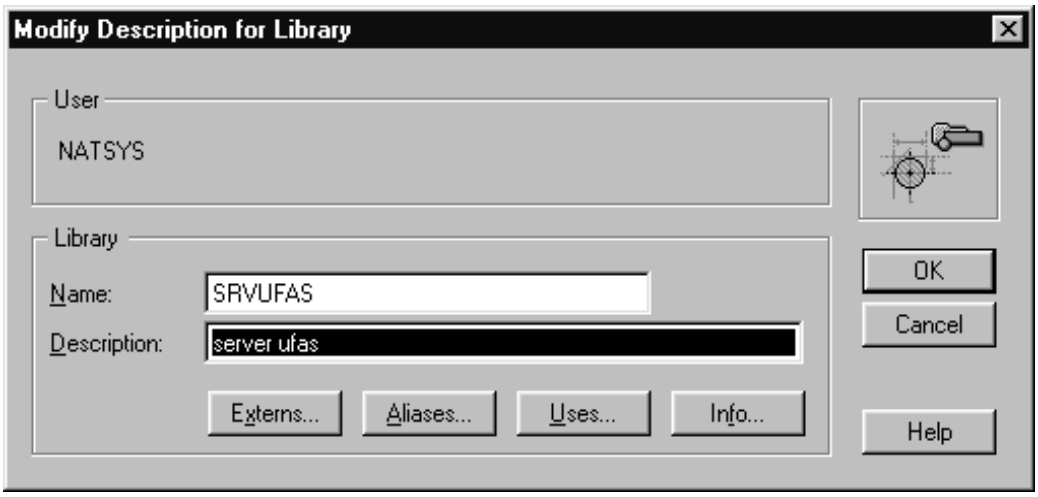

Figure F-2. Modify Library Description

Modifying External Representations

The following window allows you to define the EXTERNS... : (push the EXTERNS button).

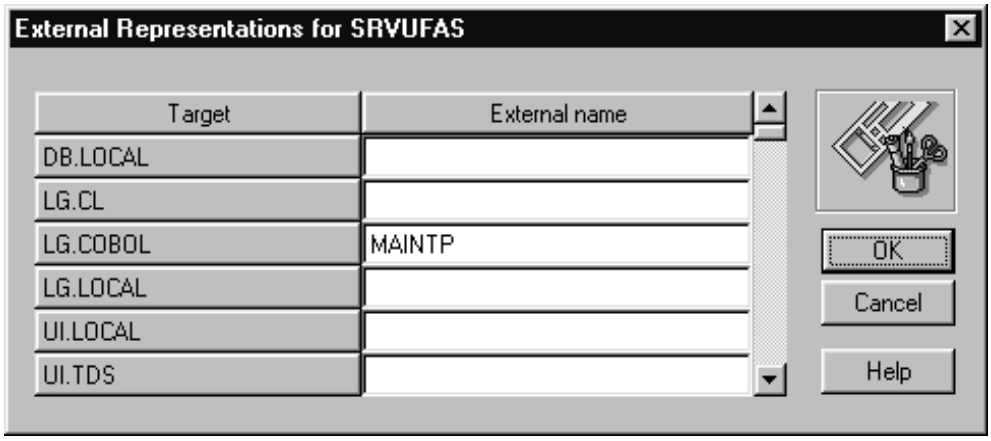

Figure F-3. Modify External Representations

You are **STRONGLY RECOMMENDED** to specify the name of the first TPR of transaction explicitly rather than leaving NATSTAR to generate this name.

This name is used in the STDS, in the MESSAGE instruction in the TRANSACTION section. The corresponding transaction name must be used in the NSCOM.INI file on the PC in the CLISERVICES section for the CPLDESTSERVICE field. The name of the COBOL source is: `N_<external name for LG.COBOL>_CBL`.

EXAMPLE:

```
EJR nscob7 lib=si7.ns7.sllib
    vl=(N_MAINTP_CBL, inlib=ns.test.sllib,
    prtlib=ns.test.lislib, culib=ns.test.culib) holdout
```

The library containing NSCOB7 delivered with ISI7 is `SI7.NS7.SLLIB.` Other libraries must be created by the user.

❑

F.2.2 Link Time

When all files are compiled successfully, link the final transaction application.

A link job is provided in the SI7.NS7.SLLIB library and can be used as example. It needs parameters which are documented inside the job source file.

EXAMPLE:

```
EJR nslk7 lib=si7.ns7.sllib
    vl = (MAINTP, lnksl=scha.sllib, outlib=scha.smlib,
          prtlib=scha.edition, culib=ns.test.culib) holdout
```

❑

F.2.3 TDS Generation

Before TDS generation, modify the STDS in three ways:

1. Add a MESSAGE instruction in the TRANSACTION section of the type:

   ```
   MESSAGE "TXUFAS" ASSIGN  TO  MAINTP
               IMPLICIT  COMMITMENT
               AUTHORITY-CODES
               ARE 1,2,3,4,5,6,7,8,9,11,12,13,14,15
   ```

2. Add an ASSIGN instruction between the UFAS controlled file and an IFN known in the client part of the application (here IFNCLI)

   ```
    13    SELECT EXTERNAL SAMPLE_UFAS ASSIGN TO IFNCLI
    14    ORGANIZATION IS UFF INDEXED
    15    ACCESS MODE IS RANDOM
    16    RECORD KEY IS NUM-C.
    17              *END
   ```

3. Set the message length to 32000.

F.2.4 Launching TDS

Add the controlled file in the job that launches your TDS then start TDS.

F.2.5 Launching the NSCOM Daemon and Pseudo Servers on OPEN 7

Edit the `nscom.ini` file on OPEN 7. This file is in the directory `/usr/natsys/ini` and belongs to ROOT user. In the SRVSERVICES block, create a service for your TDS with the following arguments:

```
Service=SCHA  <name of the argument DESTSERVICE in the CLISERVICES
  block in the NSCOM.INI file on the PC client>
    CplService       =
    ProcessId        =/usr/natsys/bin/nserv7 -t scha -d -c 10
     (The -t option is followed by the name of the launched TDS)
    Start            =AUTO
   The pseudo server will be launched automatically when the nsserver is started)
    SrvTimeout       =-1
    SrvAValidDelay   =-1
    WorkDirectory    =
    Trace            =NO
    MinSessLimits    =2     (The value 2 is mandatory)
    MaxSessLimits    =32
```

When this is done, set the mandatory environment variables by submitting the following command:

```
cd /usr/natsys/ini
    . ./inivar
```

You can then start the NSCOM daemon `nsserver` with the following command:

```
cd /usr/natsys
nsserver -start
```

The server is started together with the two NSERV7 pseudo-servers.

When all the servers are running, you can launch the client application on the PC side.

NOTE:
In NSCOM.INI, in the [TCPIP] section, set MaxBufSize = 32 000.

F.2.6 Editing the NSCOM.INI File on PC Client

Add the following lines in the CLISERVICES section of the NSCOM.INI file

```
Service =SRV_UFAS
```
(name of logical service in THINGS_SET_DEFAULT_SERVER)

```
CplService      =
RemoteNode      = bc06-41
```
(name of the OPEN 7 where nserv7 is running)

```
PortNumber      = 2000
```
(port number on which NSSERVER is listening on OPEN 7 and as defined in the NSCOM.INI of OPEN 7, refer to the NatStar TP documentation)

```
DestService     = SCHA
```
(same as SERVICE = in SRVSERVICES of the OPEN 7 NSCOM.INI file)

```
CplDestService  = TXUFAS
```
(name of the transaction specified in the MESSAGE instruction in the STDS TRANSACTION section)

```
SessionType     = TCPIP
```
(the transport protocol used)

F.2.7 Launching the PC Client Application

Enter the NatStar application and click on the RUN icon.

Enter your own USER, PASSWD, PROJECT, BILLING and click on the CONNECTION push-button.

When you are connected, after checking the connected message:

1. Access UFAS file

   In the N° Count field, type a count number with 6 digits (e.g.: 000001), and click the Read File push-button.

   You should obtain "JACQUES DUPOND" in the result field.

2. CONCAT function

   Type any strings in the entry fields, string1 and string2 like "TITI" and "TOTO", click the Concat push-button.

   You should obtain "TITITOTO" in the result field.

You should obtain the next screen:

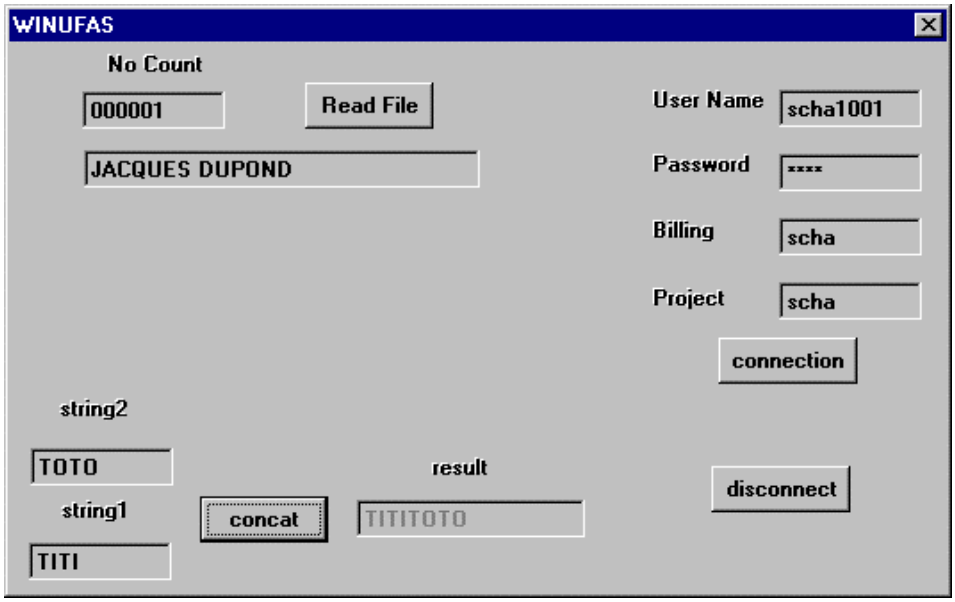

Figure F-4. Operational Connected Screen Aspect of the Application TESTUFAS

To disconnect from the TDS, click on the disconnect push-button, and check the DISCONNECT message.

To close the window click on the end application icon (X) on the top right of the screen.

G. NCL Sequence Listings

The following are the full listings are of the NCL sequences shown only partially in the screens in Appendix B *Developing the CONCAT Server Application*.

G.1 Enter NCL for the SRVUFAS Library (Server Part)

```
@MODE GRAMMAR
; File exported by NATSYS on 29/05/97,19:05:21
; File exported with IMEX_IMEX.DLL dated on Jan 14 1997 version 1080
; Resources exported:
;     L.SRVUFAS
; Options:
;     EXTERNALS=ON
;     DESCRIPTIONS=ON
;     DOCS=ON
;     ALIAS AND KEYWORDS=ON
;     USES=OFF
;     ZEROSIZED=ON
;     CLASS WITH METHODDEF=OFF
;     WITH METHOD IMPLEMENTATION=ON
;     ONLY ON KEYWORD=OFF
;
;---------------Beginning of Library SRVUFAS ---------------------
LIBRARY SRVUFAS
        DESCR = 'server ufas'
        DOC 0
        ENDDOC
        FUNCTION SRV_CONC
                DESCR = 'concatene un chaine de caractere'
                RETURN = INT(4)
                REMOTE = YES
                DOC 0
                ENDDOC
                PARAMETERS 3
CHAIN1& STRING& 32& & I& -& 1er parametre
CHAIN2& STRING& 32& & I& -& 2eme parametre
RES_CHAIN& STRING& 64& & -& O& resultat
                ENDPARAMETERS
```

```
                IMPLEM 10
local string loc_p1$(32)
local string loc_p2$(32)
local string loc_res$(64)

move chain1 to loc_p1$
move chain2 to loc_p2$

loc_res$ = loc_p1$&loc_p2$

move loc_res$ to res_chain
                ENDIMPLEM
        ENDFUNCTION
        FUNCTION SRV_GET_UFAS
                DESCR = 'fonction acces UFAS'
                RETURN = INT(4)
                REMOTE = YES
                DOC 0
                ENDDOC
                PARAMETERS 4
NO_COMPT& STRING& 6& & I& -& numero de compte
NAME& STRING& 15& & -& O& nom du client
NUM_ERR& INT& 4& & -& O& num d'erreur
ERR_STR& STRING& 32& & -& O& string d'erreur
                ENDPARAMETERS
                IMPLEM 37
local char clirec$(32)
Local char l_num_compt$(32)
Local string l_name_compt$(15)
Local int l_err_num%
Local string l_err_str$(32)
local char int_name$(8)
local pointer pfhandle
local char keypos$(4),char keylen$(4),char fmode$(4)
local segsiz%(2)
local pointer ptclir
local pointer ptkey

Move no_compt to l_num_compt$
move "IFNCLI" to int_name$
move " " to keypos$
move " " to keylen$
move " " to fmode$
;move sizeof art_client to segsiz%
move 32 to segsiz%
move @clirec$ to ptclir
move @l_num_compt$ to ptkey


nsufas_get_file_handle pfhandle,int_name$
```

```
nsufas_read_indexed pfhandle,ptclir,segsiz%,ptkey


Move clirec$(10..25) to name
l_err_num% = 0
l_err_str$ = "default_error"
Move l_err_num% to num_err
Move l_err_str$ to err_str

                ENDIMPLEM
        ENDFUNCTION
ENDLIBRARY
;-------------- End of Library SRVUFAS ---------------------------
```

G.2 Enter NCL for the TESTUFAS Library (Client Part)

```
@MODE GRAMMAR
; File exported by NATSYS on 29/05/97,19:05:21
; File exported with IMEX_IMEX.DLL dated on Jan 14 1997 version 1080
; Resources exported:
;     L.TESTUFAS
; Options:
;     EXTERNALS=ON
;     DESCRIPTIONS=ON
;     DOCS=ON
;     ALIAS AND KEYWORDS=ON
;     USES=OFF
;     ZEROSIZED=ON
;     CLASS WITH METHODDEF=OFF
;     WITH METHOD IMPLEMENTATION=ON
;     ONLY ON KEYWORD=OFF
;
;---------- Beginning of Library TESTUFAS ------------------------
LIBRARY TESTUFAS
        DESCR = 'client ufas'
        USES = SRVUFAS
        DOC 0
        ENDDOC
        WINDOW WINUFAS
                DESCR = 'window ufas'
                MODEL = DIALOG
                DOC 0
                ENDDOC
                LOG 143
STARUNITS
SETHEADER DIALOG WINUFAS 'WINUFAS' 148 117 500 240 PSHELLP TITLEBAR DLGBORDER
  SIZEREDRAW SAVEBITS DEFICON '' NULL NULL NULL NULL NULL IMAGE 0
ADDCTRL ENTRY NAME_EF NULL NULL NULL 363 183 80 31 1 8 MARGIN AUTOSCROLL '' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY PASS_EF NULL NULL NULL 364 151 80 31 1 8 MARGIN AUTOSCROLL '' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY BILLING_EF NULL NULL NULL 365 110 80 31 1 8 MARGIN AUTOSCROLL ''
  '' NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY PROJECT_EF NULL NULL NULL 365 73 80 31 1 8 MARGIN AUTOSCROLL '' ''
   NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY RESCON_EF NULL NULL NULL 317 37 169 31 1 8 MARGIN AUTOSCROLL '' ''
   NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL PUSHBUTTON PB_DISCON NULL 'disconnect' 261 79 78 26 1 FONT
   'MS Sans Serif' 8 BOLD
ADDEVENT PB_DISCON EXECUTED
local cstring user$(20),cstring pass$(20),cstring proj$(20),
      cstring bill$(20);
local char  comarea$(1024),char  comarecv$(1024);
local cstring servername$(80), cstring errormsg$(255),cstring srvcpl$;
```

```
local int winh%,pointer phab%;
local cstring srvcpl$;
local int ret%(4),int questid%,int comsiz%,int comsizrcv%;

winh% = 0;
phab% = 0;
srvcpl$ = "";
ret% = 0;

move string name_ef to user$;
move string pass_ef to pass$;
move string billing_ef to bill$;
move string project_ef to proj$;

;ret% = NSC_CLI_INIT%;

;IF ret% <> NSC_OK%
; message "NSC_CLI_INIT_ERR :", NSC_COM_ERROR$ (ret%,255);
; EXIT
;ENDIF
servername$ = "SRV_UFAS"
ret% = NSC_CLI_START_TP%(servername$,srvcpl$);
ret% = NSC_OK%
comarea$ = "GCOS7-logoff"
message "SENDING " ,"comarea = "&comarea$
;comsiz% = length comarea$
comsiz% = 12
ret% = NSC_CLI_EXCHANGE%(servername$,srvcpl$,winh%,phab%,comsiz%,@comarea$,
           questid%,comsizrcv%)
IF ret% <> NSC_OK%
 message "NSC_CLI_EXCHANGE_ERROR",NSC_COM_ERROR$ (ret%,255);
ELSE
  ret% = NSC_CLI_GET_A% (questid%,comsizrcv%,@comarecv$)
   IF ret% <> NSC_OK%
     message "NSC_CLI_GET_A_ERROR","ret = "& string ret%
   ELSE
     rescon_ef = comarecv$
   ENDIF

ENDIF
ret% = NSC_CLI_STOP_TP%(servername$,srvcpl$);
;ret% = NSC_CLI_TERMINATE%
~~
ADDCTRL ENTRY CHAIN1_EF NULL NULL NULL 28 86 80 31 1 8 MARGIN AUTOSCROLL '' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY CHAIN2_EF NULL NULL NULL 141 86 80 31 1 8 MARGIN AUTOSCROLL '' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY RES_EF NULL NULL NULL 29 45 80 31 1 8 MARGIN AUTOSCROLL '' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL PUSHBUTTON PB_CONC NULL 'concat' 150 42 72 26 1 FONT
```

```
     'MS Sans Serif' 8 BOLD
ADDEVENT PB_CONC EXECUTED
local string l_p1$(32)
local string l_p2$(32)
local string l_res$(64)
local int ret%(4)

move chain1_ef to l_p1$
move chain2_ef to l_p2$

ret% = srv_conc ( l_p1$,l_p2$,l_res$)

move l_res$ to res_ef
~~
ADDCTRL PUSHBUTTON PB_CONNECT NULL 'connection' 262 178 78 26 1 FONT
  'MS Sans Serif' 8 BOLD
ADDEVENT PB_CONNECT EXECUTED
local cstring user$(20),cstring pass$(20),cstring proj$(20),cstring bill$(20);
local char  comarea$(1024),char  comarecv$(1024);
local cstring servername$(80), cstring errormsg$(255),cstring srvcpl$;

local int winh%,pointer phab%;
local cstring srvcpl$;
local int ret%(4),int questid%,int comsiz%,int comsizrcv%;

winh% = 0;
phab% = 0;
srvcpl$ = "";
ret% = 0;

move string name_ef to user$;
move string pass_ef to pass$;
move string billing_ef to bill$;
move string project_ef to proj$;

;ret% = NSC_CLI_INIT%;

;IF ret% <> NSC_OK%
; message "NSC_CLI_INIT_ERR :", NSC_COM_ERROR$ (ret%,255);
; EXIT
;ENDIF
servername$ = "SRV_UFAS"
ret% = NSC_CLI_START_TP%(servername$,srvcpl$);

ret% = NSC_OK%
comarea$ = "GCOS7-logon -u "&user$&" -p "&pass$&" -r "&proj$&" -b "&bill$
message "SENDING " ,"comarea = "&comarea$
comsiz% = length comarea$
;comsiz% = 47
ret% = NSC_CLI_EXCHANGE%(servername$,srvcpl$,winh%,phab%,comsiz%,@comarea$,
    questid%,comsizrcv%)
```

```
IF ret% <> NSC_OK%
 message "NSC_CLI_EXCHANGE_ERROR",NSC_COM_ERROR$ (ret%,255);
ELSE
  ret% = NSC_CLI_GET_A% (questid%,comsizrcv%,@comarecv$)
   IF ret% <> NSC_OK%
     message "NSC_CLI_GET_A_ERROR","ret = "& string ret%
   ELSE
     rescon_ef = comarecv$
   ENDIF

ENDIF
ret% = NSC_CLI_STOP_TP%(servername$,srvcpl$);

;ret% = NSC_CLI_TERMINATE%
~~
ADDCTRL ENTRY USER_EF NULL NULL NULL 105 128 208 31 1 8 MARGIN AUTOSCROLL
  DISABLE '' '' NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY NO_COMPT_EF NULL NULL NULL 149 166 80 31 1 8 MARGIN AUTOSCROLL ''
  '' NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL TEXT COMPT_TXT NULL 'No de compte' 51 166 80 21 1 8 ERASERECT LEFT OVER
   FONT 'MS Sans Serif' 8 BOLD
ADDCTRL PUSHBUTTON PB_LISTE NULL 'read file' 11 128 72 26 1 FONT
  'MS Sans Serif' 8 BOLD
ADDEVENT PB_LISTE EXECUTED
local string loc_name$(15)
local string loc_no_compt$(6)
local int loc_error_num%
local string loc_err_str$(32)
local int loc_ret%

move no_compt_ef to loc_no_compt$

loc_ret% = srv_get_ufas (loc_no_compt$,loc_name$,loc_error_num%,loc_err_str$)


move loc_name$ to user_ef
~~
CHGEVENT NULL INIT
things_set_default_server "SRV_UFAS"
~~
                ENDLOG
        ENDWINDOW
ENDLIBRARY
;------------- End of Library TESTUFAS --------------------------
```

G.3 Connection Template

```
LIBRARY G7USRPASS
        DESCR = 'library global de user password'
        DOC 0
        ENDDOC
        TEMPLATE G7CONECT
                DESCR = 'connection to GCOS7'
                MODEL = TEMPLATE
                DOC 0
                ENDDOC
                LOG 81
STARUNITS
SETHEADER CONTROLTPL G7CONECT 159 210 176 197 '' NULL NULL SIZEABLE
ADDCTRL TEXT NAME_TXT NULL 'User Name' 10 154 63 21 1 8 ERASERECT LEFT OVER
  FONT 'MS Sans Serif' 8 BOLD
ADDCTRL TEXT PASS_TXT NULL 'Password' 10 117 55 21 1 8 ERASERECT LEFT OVER
  FONT 'MS Sans Serif' 8 BOLD
ADDCTRL TEXT BILL_TXT NULL 'Billing' 10 80 35 21 1 8 ERASERECT LEFT OVER
  FONT 'MS Sans Serif' 8 BOLD
ADDCTRL TEXT PROJ_TXT NULL 'Project' 10 43 41 21 1 8 ERASERECT LEFT OVER
  FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY NAME_EF NULL NULL PASS_EF 83 152 80 12 1 8 MARGIN AUTOSCROLL
  REQUIRED NOBLANKS '''A''..''Z'',''a''..''z'',''0''..''9'',''_'',''-''' ''
  NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY PASS_EF NULL NULL BILL_EF 83 116 80 12 1 8 MARGIN AUTOSCROLL
 REQUIRED NOBLANKS HIDETEXT '''A''..''Z'',''a''..''z'',''0''..''9'',''_'',''-'''
 '' NONE LEFT BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY BILL_EF NULL NULL PROJ_EF 83 80 80 12 1 8 MARGIN AUTOSCROLL
  NOBLANKS '''A''..''Z'',''a''..''z'',''0''..''9'',''_'',''-''' '' NONE LEFT
 BELOW FONT 'MS Sans Serif' 8 BOLD
ADDCTRL ENTRY PROJ_EF NULL NULL NULL 84 44 80 12 1 8
   MARGIN AUTOSCROLL NOBLANKS
  '''A''..''Z'',''a''..''z'',''0''..''9'',''_'',''-''' '' NONE LEFT BELOW
  FONT 'MS Sans Serif' 8 BOLD
ADDCTRL PUSHBUTTON CONNECT NULL 'connection' 48 7 78 26 1 FONT
 'MS Sans Serif' 8 BOLD
ADDEVENT CONNECT INIT
move string '' to bill_ef
move string '' to proj_ef
~~
ADDEVENT CONNECT EXECUTED
local cstring user$(20),cstring pass$(20),cstring proj$(20),cstring bill$(20);
local char  comarea$(128),char  comarecv$(128);
local cstring servername$(80), cstring errormsg$(255),cstring srvcpl$;

local int winh%,pointer phab%;
local cstring srvcpl$;
local int ret%(4),int questid%,int comsiz%,int comsizrcv%;
```

```
winh% = 0;
phab% = 0;
srvcpl$ = "";
ret% = 0;

move string name_ef to user$;
move string pass_ef to pass$;
move string bill_ef to bill$;
move string proj_ef to proj$;



servername$ = "NOM_DU_SERVEUR_LOGIQUE"
ret% = NSC_OK%
ret% = NSC_CLI_START_TP%(servername$,srvcpl$);
IF ret% <> NSC_OK%
 message "NSC_CLI_START_TP ERROR=",NSC_COM_ERROR$ (ret%,255);
ELSE
 ret% = NSC_OK%

 IF proj$ <> ''
  IF bill$ <> ''
    comarea$ = "GCOS7-logon -u "&user$&" -p "&pass$&" -r "&proj$&" -b "&bill$
  ELSE
    comarea$ = "GCOS7-logon -u "&user$&" -p "&pass$&" -r "&proj$
  ENDIF
 ELSE
  IF bill$ <> ''
   comarea$ = "GCOS7-logon -u "&user$&" -p "&pass$&" -b "&bill$
  ELSE
   comarea$ = "GCOS7-logon -u "&user$&" -p "&pass$
  ENDIF
 ENDIF
 ;message "COMAREA",comarea$
 comsiz% = length comarea$
 ret% = NSC_CLI_EXCHANGE%(servername$,srvcpl$,winh%,phab%,comsiz%,@comarea$,
  questid%,comsizrcv%)
 IF ret% <> NSC_OK%
  message "NSC_CLI_EXCHANGE_ERROR",NSC_COM_ERROR$ (ret%,255);
 ELSE
   ret% = NSC_CLI_GET_A% (questid%,comsizrcv%,@comarecv$)
    IF ret% <> NSC_OK%
      message "NSC_CLI_GET_A_ERROR",NSC_COM_ERROR$ (ret%,255)
    ELSE
      comarecv$(comsizrcv%) = 0
          errormsg$ = comarecv$
      message "RETURN CONNECT",errormsg$

    ENDIF
```

```
 ENDIF
 ret% = NSC_CLI_STOP_TP%(servername$,srvcpl$);
 IF ret% <> NSC_OK%
  message "NSC_CLI_STOP_TP ERROR=",NSC_COM_ERROR$ (ret%,255);
 ENDIF

ENDIF
~~
                ENDLOG
        ENDTEMPLATE
        TEMPLATE G7DISCNT
                DESCR = 'disconnection from GCOS7'
                MODEL = TEMPLATE
                DOC 0
                ENDDOC
                LOG 49
STARUNITS
SETHEADER CONTROLTPL G7DISCNT 159 210 179 44 '' NULL NULL SIZEABLE
ADDCTRL PUSHBUTTON DISCN_PB NULL 'disconnect' 47 10 78 26 1 FONT
  'MS Sans Serif' 8 BOLD
ADDEVENT DISCN_PB EXECUTED
local cstring user$(20),cstring pass$(20),cstring proj$(20),
      cstring bill$(20);
local char  comarea$(128),char  comarecv$(128);
local cstring servername$(80), cstring errormsg$(255),cstring srvcpl$;

local int winh%,pointer phab%;
local cstring srvcpl$;
local int ret%(4),int questid%,int comsiz%,int comsizrcv%;

winh% = 0;
phab% = 0;
srvcpl$ = "";
ret% = 0;


servername$ = "NOM_DU_SERVEUR_LOGIQUE"
ret% = NSC_CLI_START_TP%(servername$,srvcpl$);
IF ret% <> NSC_OK%
 message "NSC_CLI_START_TP ERROR=",NSC_COM_ERROR$ (ret%,255);
ELSE
 ret% = NSC_OK%
 comarea$ = "GCOS7-logoff"
 ;message "SENDING " ,"comarea = "&comarea$
 comsiz% = length comarea$
 ret% = NSC_CLI_EXCHANGE%(servername$,srvcpl$,winh%,phab%,comsiz%,@comarea$,
  questid%,comsizrcv%)
 IF ret% <> NSC_OK%
  message "NSC_CLI_EXCHANGE_ERROR",NSC_COM_ERROR$ (ret%,255);
```

```
 ELSE
   ret% = NSC_CLI_GET_A% (questid%,comsizrcv%,@comarecv$)
    IF ret% <> NSC_OK%
      message "NSC_CLI_GET_A_ERROR", NSC_COM_ERROR$ (ret%,255);
    ELSE
      message "OK ","Disconnected from Server"
   ENDIF

 ENDIF

ret% = NSC_CLI_STOP_TP%(servername$,srvcpl$);
if ret% <> NSC_OK%
   message "NSC_CLI_STOP_TP ERROR=", NSC_COM_ERROR$ (ret%,255)
endif
ENDIF
exit
~~
                ENDLOG
        ENDTEMPLATE
ENDLIBRARY
```

H. NatStar TP Error Codes

These errors can be appear in the execution of the generated NATSTAR MAIN PROGRAM:

For TDS, these errors appear in the trace file in <tdsname>.DEBUG file (do not forget to put the NATSTAR session in trace mode). See also in *NATSTAR/TP User Manual* the chapter *NATSTAR/TP Error Messages*.

The errors NTxx listed here correspond to the errors NST-5xx in the *NATSTAR/TP User Manual* the chapter *NATSTAR/TP Error Messages.*

```
NT00 : NST-000 NO ERROR
NT01 : NST-501 ** INVALID_TRANSAC_NAME
NT02 : NST-502 ** CANNOT_ALLOCATE_MEMORY
NT03 : NST-503 ** UNKNOWN_INFO_VALUE
NT04 : NST-504 ** NO_HANDLE
NT05 : NST-505 ** INVALID_INFO_SIZE
NT06 : NST-506 ** INVALID_PARAMETER
NT07 : NST-507 ** INVALID_ARRAY_TYPE
NT08 : NST-508 ** INVALID_ARRAY_SIZE
NT09 : NST-509 ** WRONG_ACK_SIZE
NT10 : NST-510 ** WRONG_ACK
NT11 : NST-511 ** SEND_WRONG_BUF_INDEX
NT12 : NST-512 ** INDEXATION_ERROR
NT13 : NST-513 ** BUFFER_OVERFLOW
NT14 : NST-514 ** TOO_MANY_BUFFER
NT15 : NST-515 ** UNKNOWN_DATAID
NT16 : NST-516 ** DATAID_ALREADY_USED
NT17 : NST-517 ** INVALID_NUM
NT18 : NST-518 ** READ_INDEX_TABLE
NT19 : NST-519 ** BAD_SYS_DATAID
NT20 : NST-520 ** UNKNOWN_USERTYPE
NT21 : NST-521 ** USERTYPE_ALREADY_USED
NT22 : NST-522 ** INVALID_USERTYPE
NT23 : NST-523 ** INVALID_USERTYPE_FUNC
NT24 : NST-524 ** BAD_TRANSACTION_HANDLE
NT25 : NST-525 ** REQUEST_IS_IN_PROGRESS
NT26 : NST-526 ** BUFFER_NOT_ALLOCATED
NT27 : NST-527 ** RECV_BAD_NUM_BUFFER
NT28 : NST-528 ** REPLY_HAS_NO_BUFFER
```

```
NT29 : NST-529 ** NOT_IMPLEMENTED
NT30 : NST-530 ** BAD_LENGTH_VALUE
NT31 : NST-531 ** INVALID_DATA_VALUE
NT32 : NST-532 ** INVALID_DATA_TYPE
NT33 : NST-533 ** INVALID_CONV_MODE
NT34 : NST-534 ** INVALID_TRAILER
NT35 : NST-535 ** INVALID_PHYS_HEADER
NT36 : NST-536 ** INVALID_LOG_HEADER
NT37 : NST-537 ** INVALID_HEADER_CMD
NT38 : NST-538 ** BAD_VERSION_NUMBER
NT39 : NST-539 ** INVALID_CONV_HEADER
NT40 : NST-540 ** INVALID_DB_TYPE
NT41 : NST-541 ** XA_INFO_MISSING
NT42 : NST-542 ** DATA_TYPE_MISMATCH
NT43 : NST-543 ** BAD_REMOTE_TRID
NT44 : NST-544 ** ROLLBACK_FAILED
NT45 : NST-545 ** COMMIT_FAILED
NT46 : NST-546 ** RESERVED_DATAID_VALUE
NT47 : NST-547 ** TOO_MANY_SRV_FUNCTIONS
NT48 : NST-548 ** READ_SYSTEM_DATA
NT49 : NST-549 ** ALREADY_INITIALIZED
NT50 : NST-550 ** NOT_INITIALIZED
NT51 : NST-551 ** CANNOT_FIND_FUNCTION
NT52 : NST-552 ** CANNOT_FIND_LIBRARY
NT53 : NST-553 ** LAYER_NOT_OPENED
NT54 : NST-554 ** CANNOT_GET_FUNC_ADDRESS
NT55 : NST-555 ** CANNOT_LOAD_DLL
NT56 : NST-556 ** ROLLED_BACK
NT57 : NST-557 ** ABEND
NT58 : NST-558 ** ABORT
NT59 : NST-559 ** SERVICE_IS_SHAREABLE
NT60 : NST-560 ** CAST_FAILED
```

New Error Messages with NSCOM Light: NS02OP7G

```
NT61 : NST-561 ** INIT_FAILED
NT62 : NST-562 ** GlobalAlloc_FAILED
NT63 : NST-563 ** GlobalLock_FAILED
NT64 : NST-564 ** SESSION_ENTRY_NOT_ALLOCATED
NT65 : NST-565 ** TOO_MANY_SESSION
NT66 : NST-566 ** UNKNOWN_VIRTUAL_NODE
NT67 : NST-567 ** CONNEXION_FAILED
NT68 : NST-568 ** DROP_DATA_FAILED
NT69 : NST-569 ** SESSION_ENTRY_ALREADY_USED
NT70 : NST-570 ** SOCKET_ERROR
NT71 : NST-571 ** NO_CONNECTION_FOR_THIS_REQUEST
NT72 : NST-572 ** SOCKET_SELECT_NOT_READABLE
NT73 : NST-573 ** SOCKET_SELECT_ERROR Code=
NT74 : NST-574 ** SOCKET_CONNECT_ERROR Code=
NT75 : NST-575 ** SOCKET_SEND_ERROR Code=
NT76 : NST-576 ** SOCKET_RECEIVE_ERROR Code=
NT77 : NST-577 ** OPEN7_GATEWAY_FAILED Reason=
NT78 : NST-578 ** GetServByName_ERROR Code=
NT79 : NST-579 ** GetHostByName_ERROR Code=
NT80 : NST-580 ** HOST_NAME_UNDEFINED
NT81 : NST-581 ** REQUEST_UNKNOWN
NT82 : NST-582 ** TOO_MANY_VIRTUAL_NODE
NT83 : NST-583 ** INVALID_VIRTUAL_NODE
NT84 : NST-584 ** CANNOT_OPEN_FILE_CNFGDSN.CFG
NT85 : NST-585 ** ERROR_DURING_CNFGDSN.CFG_FILE_ACCESS
NT86 : NST-586 ** USER_PASSWORD_NOT_DEFINED
NT87 : NST-587 ** UNKNOWN_ERROR
```

I. Error Codes for the Socket Interface on PC

The following is a list of possible error codes returned by the WSAGetLastError call, along with their extended explanations. Errors are listed in alphabetical order by error macro. Some error codes defined in WINSOCK2.H are not returned from any function - these have not been listed here.

WSAEACCES

```
(10013)
Permission denied.
```

An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for sendto without broadcast permission being set using setsockopt (SO_BROADCAST).

WSAEADDRINUSE

```
(10048)
Address already in use.
```

Only one usage of each socket address (protocol/IP address/port) is normally permitted. This error occurs if an application attempts to bind a socket to an IP address/port that has already been used for an existing socket, or a socket that was not closed properly, or one that is still in the process of closing. For server applications that need to bind multiple sockets to the same port number, consider using setsockopt (SO_REUSEADDR). Client applications usually need not call bind at all - connect will choose an unused port automatically.

WSAEADDRNOTAVAIL

```
(10049)
Cannot assign requested address.
```

The requested address is not valid in its context. Normally results from an attempt to bind to an address that is not valid for the local machine, or connect/sendto an address or port that is not valid for a remote machine (e.g. port 0).

WSAEAFNOSUPPORT

```
(10047)
Address family not supported by protocol family.
```

An address incompatible with the requested protocol was used. All sockets are created with an associated "address family" (i.e. AF_INET for Internet Protocols) and a generic protocol type (i.e. SOCK_STREAM). This error will be returned if an incorrect protocol is explicitly requested in the socket call, or if an address of the wrong family is used for a socket, e.g. in sendto.

WSAEALREADY

```
(10037)
Operation already in progress.
```

An operation was attempted on a non-blocking socket that already had an operation in progress - i.e. calling connect a second time on a non-blocking socket that is already connecting, or canceling an asynchronous request (WSAAsyncGetXbyY) that has already been canceled or completed.

WSAECONNABORTED

```
(10053)
Software caused connection abort.
```

An established connection was aborted by the software in your host machine, possibly due to a data transmission timeout or protocol error.

WSAECONNREFUSED

```
(10061)
Connection refused.
```

No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host - i.e. one with no server application running.

WSAECONNRESET

```
(10054)
Connection reset by peer.
```

An existing connection was forcibly closed by the remote host. This normally happens if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host used a "hard close" (see setsockopt for more information on the SO_LINGER option on the remote socket.)

WSAEDESTADDRREQ

```
(10039)
Destination address required.
```

A required address was omitted from an operation on a socket. For example, this error will be returned if sendto is called with the remote address of ADDR_ANY.

WSAEFAULT

```
(10014)
Bad address.
```

The system detected an invalid pointer address in attempting to use a pointer argument of a call. This error occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of an argument which is a struct sockaddr is smaller than sizeof(struct sockaddr).

WSAEHOSTDOWN

```
(10064)
Host is down.
```

A socket operation failed because the destination host was down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.

WSAEHOSTUNREACH

```
(10065)
No route to host.
```

A socket operation was attempted to an unreachable host. See WSAENETUNREACH.

WSAEINPROGRESS

```
(10036)
Operation now in progress.
```

A blocking operation is currently executing. Windows Sockets only allows a single blocking operation to be outstanding per task (or thread), and if any other function call is made (whether or not it references that or any other socket), the function fails with the WSAEINPROGRESS error.

WSAEINTR

```
(10004)
Interrupted function call.
```

A blocking operation was interrupted by a call to WSACancelBlockingCall.

WSAEINVAL

```
(10022)
Invalid argument.
```

Some invalid argument was supplied (for example, specifying an invalid level to the setsockopt function). In some instances, it also refers to the current state of the socket - for instance, calling accept on a socket that is not listening.

WSAEISCONN

```
(10056)
Socket is already connected.
```

A connect request was made on an already connected socket. Some implementations also return this error if sendto is called on a connected SOCK_DGRAM socket (For SOCK_STREAM sockets, the to parameter in sendto is ignored), although other implementations treat this as a legal occurrence.

WSAEMFILE

```
(10024)
Too many open files.
```

Too many open sockets. Each implementation may have a maximum number of socket handles available, either globally, per process, or per thread.

WSAEMSGSIZE

```
(10040)
Message too long.
```

A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram was smaller than the datagram itself.

WSAENETDOWN

```
(10050)
Network is down.
```

A socket operation encountered a dead network. This could indicate a serious failure of the network system (i.e. the protocol stack that the WinSock DLL runs over), the network interface, or the local network itself.

WSAENETRESET

```
(10052)
Network dropped connection on reset.
```

The host you were connected to crashed and rebooted. May also be returned by setsockopt if an attempt is made to set SO_KEEPALIVE on a connection that has already failed.

WSAENETUNREACH

```
(10051)
Network is unreachable.
```

A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.

WSAENOBUFS

```
(10055)
No buffer space available.
```

An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

WSAENOPROTOOPT

```
(10042)
Bad protocol option.
```

An unknown, invalid or unsupported option or level was specified in a getsockopt or setsockopt call.

WSAENOTCONN

```
(10057)
Socket is not connected.
```

A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using sendto), no address was supplied. Any other type of operation might also return this error - for example, setsockopt setting SO_KEEPALIVE if the connection has been reset.

WSAENOTSOCK

```
(10038)
Socket operation on non-socket.
```

An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for select, a member of an fd_set was not valid.

WSAEOPNOTSUPP

```
(10045)
Operation not supported.
```

The attempted operation is not supported for the type of object referenced. Usually this occurs when a socket descriptor to a socket that cannot support this operation, for example, trying to accept a connection on a datagram socket.

WSAEPFNOSUPPORT

```
(10046)
Protocol family not supported.
```

The protocol family has not been configured into the system or no implementation for it exists. Has a slightly different meaning to WSAEAFNOSUPPORT, but is interchangeable in most cases, and all Windows Sockets functions that return one of these specify WSAEAFNOSUPPORT.

WSAEPROCLIM

```
(10067)
Too many processes.
```

A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously. WSAStartup may fail with this error if the limit has been reached.

WSAEPROTONOSUPPORT

```
(10043)
Protocol not supported.
```

The requested protocol has not been configured into the system, or no implementation for it exists. For example, a socket call requests a SOCK_DGRAM socket, but specifies a stream protocol.

WSAEPROTOTYPE

```
(10041)
Protocol wrong type for socket.
```

A protocol was specified in the socket function call that does not support the semantics of the socket type requested. For example, the ARPA Internet UDP protocol cannot be specified with a socket type of SOCK_STREAM.

WSAESHUTDOWN

```
(10058)
Cannot send after socket shutdown.
```

A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call. By calling shutdown a partial close of a socket is requested, which is a signal that sending or receiving or both has been discontinued.

WSAESOCKTNOSUPPORT

```
(10044)
Socket type not supported.
```

The support for the specified socket type does not exist in this address family. For example, the optional type SOCK_RAW might be selected in a socket call, and the implementation does not support SOCK_RAW sockets at all.

WSAETIMEDOUT

```
(10060)
Connection timed out.
```

A connection attempt failed because the connected party did not properly respond after a period of time, or an established connection failed because connected host has failed to respond.

WSAEWOULDBLOCK

```
(10035)
Resource temporarily unavailable.
```

This error is returned from operations on non-blocking sockets that cannot be completed immediately, for example, recv when no data is queued to be read from the socket. It is a non-fatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect on a non-blocking SOCK_STREAM socket, since some time must elapse for the connection to be established.

WSAHOST_NOT_FOUND

```
(11001)
Host not found.
```

No such host is known. The name is not an official hostname or alias, or it cannot be found in the database(s) being queried. This error may also be returned for protocol and service queries, and means the specified name could not be found in the relevant database.

WSA_INVALID_HANDLE

```
(OS dependent)
Specified event object handle is invalid.
```

An application attempts to use an event object, but the specified handle is not valid.

WSA_INVALID_PARAMETER

```
(OS dependent)
One or more parameters are invalid.
```

An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a problem with one or more parameters.

WSAINVALIDPROCTABLE

```
(OS dependent)
Invalid procedure table from service provider.
```

A service provider returned a bogus proc table to WS2_32.DLL. (Usually caused by one or more of the function pointers being NULL.)

WSAINVALIDPROVIDER

```
(OS dependent)
Invalid service provider version number.
```

A service provider returned a version number other than 2.0.

WSA_IO_PENDING

```
(OS dependent)
Overlapped operations will complete later.
```

The application has initiated an overlapped operation which cannot be completed immediately. A completion indication will be given at a later time when the operation has been completed.

WSA_IO_INCOMPLETE

```
(OS dependent)
Overlapped I/O event object not in signaled state.
```

The application has tried to determine the status of an overlapped operation which is not yet completed. Applications that use WSAWaitForMultipleEvents in a polling mode to determine when an overlapped operation has completed, will get this error code until the operation is complete.

WSA_NOT_ENOUGH_MEMORY

```
(OS dependent)
Insufficient memory available.
```

An application used a Windows Sockets function which directly maps to a Win32 function. The Win32 function is indicating a lack of required memory resources.

WSANOTINITIALISED

```
(10093)
Successful WSAStartup not yet performed.
```

Either the application has not called WSAStartup or WSAStartup failed. The application may be accessing a socket which the current active task does not own (i.e. trying to share a socket between tasks), or WSACleanup has been called too many times.

WSANO_DATA

```
(11004)
Valid name, no data record of requested type.
```

The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a hostname -> address translation attempt (using gethostbyname or WSAAsyncGetHostByName) which uses the DNS (Domain Name Server), and an MX record is returned but no A record - indicating the host itself exists, but is not directly reachable.

WSANO_RECOVERY

```
(11003)
This is a non-recoverable error.
```

This indicates some sort of non-recoverable error occurred during a database lookup. This may be because the database files (e.g. BSD-compatible HOSTS, SERVICES or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.

WSAPROVIDERFAILEDINIT

```
(OS dependent)
Unable to initialize a service provider.
```

Either a service provider's DLL could not be loaded (LoadLibrary failed) or the provider's WSPStartup/NSPStartup function failed.

WSASYSCALLFAILURE

```
(OS dependent)
System call failure.
```

Returned when a system call, that should never fail, does. For example, if a call to WaitForMultipleObjects fails or one of the registry functions fails trying to manipulate the protocol/namespace catalogs.

WSASYSNOTREADY

```
(10091)
Network subsystem is unavailable.
```

This error is returned by WSAStartup if the Windows Sockets implementation cannot function at this time because the underlying system it uses to provide network services is currently unavailable. Users should check:

- that the appropriate Windows Sockets DLL file is in the current path.
- that they are not trying to use more than one Windows Sockets implementation simultaneously. If there is more than one WINSOCK DLL on your system, be sure the first one in the path is appropriate for the network subsystem currently loaded.
- the Windows Sockets implementation documentation to be sure all necessary components are currently installed and configured correctly.

WSATRY_AGAIN

```
(11002)
Non-authoritative host not found.
```

This is usually a temporary error during hostname resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.

WSAVERNOTSUPPORTED

```
(10092)
WINSOCK.DLL version out of range.
```

The current Windows Sockets implementation does not support the Windows Sockets specification version requested by the application. Check that no old Windows Sockets DLL files are being accessed.

WSAEDISCON

```
(10094)
Graceful shutdown in progress.
```

Returned by recv, WSARecv to indicate the remote party has initiated a graceful shutdown sequence.

WSA_OPERATION_ABORTED

```
(OS dependent)
Overlapped operation aborted.
```

An overlapped operation was canceled due to the closure of the socket, or the execution of the SIO_FLUSH command in WSAIoctl.

Glossary

A

API
Application Programming Interface.

B

Bull/NatStar
The Bull product offering that consists of NatStar with GCOS 7 extensions. These extensions provide: 1) code generation for GCOS 7; and 2) NSCOM7, NSERV7, and PSL7 which is a GCOS 7 implementation of the NatStar Communication Services middleware.

C

Client
A client is an application that interacts with server application(s). Sometimes called **client application**.

Client system
A client system or client workstation (the terms are synonymous) is either:
1. a PC that operates under Windows or OS/2; or
2. a workstation that operates under UNIX. A client system executes the client portion of a client/server application.

Client workstation
See Client system.

Communication services
Communication services is the generic name for the software (middleware) that handles the communication between the client and the server in a client/server environment.

Conventional TPR
A conventional TPR does *not* use NSCOM7. Typically a conventional TPR is developed using methods such as programming in COBOL-85.

D

Developers
Developers are those who are responsible for implementing client/server applications. Developers may generate applications using NatStar, and/or may use other application development methods.

G

GUI
Graphical User Interface.

N

NatStar
NatStar is a toolset that supports visually oriented application development.

NatStar Communication Services
NatStar Communication Services is the name used to refer to the entire collection of modules that make up the RPC, runtime, and communication support for client/server applications. This collection of software is also referred to as middleware.

NatStar development workstation
The NatStar development workstation is a PC or UNIX workstation on which the NatStar software is installed.

Networking services
Networking services is the generic name for the data communications software and hardware that handle the transmission of messages between the client system and the server system in a client/server configuration.

NSCOM7
NSCOM7 is the GCOS 7/OPEN 7 implementation of the NatStar Communication Services.

NSCOM7 TPR
An NSCOM7 TPR uses NSCOM7 for interaction with clients. Typically an NSCOM7 TPR is developed using NatStar.

NSERV7
nserv7 is a pseudo server running on OPEN 7 and associated with a single TDS.

R

RPC
Remote Procedure Call.

S

SCB
Sequence Control Block

Server
A server is an application that interacts with clients. Sometimes called **server application**.

Server system
A server system runs under GCOS 7, and executes the server portion of a client/server application.

Service
Service is a synonym for server [application]. This term is used in the documentation provided by Nat Systems.

System Administrators
System Administrators are those who are responsible for the configuration, installation, and administration of a Bull/NatStar client/server environment or some subset of such an environment.

U

Users
Users are those who use client applications to carry out their work. Generally, users are not developers. (They are sometimes called "end users," but that term is not used in this document.)

Index

A

accessing databases
  during simulated testing 5-102
  from conventional subroutine 5-8, 5-11
API
  NS02OP7G 8-3
application
  compiling F-3
  linking F-5

B

BATCH
  environment 6-1
  environment with ORACLE 6-7
  NatStar-generated code 2-16
benefits of NatStar 1-3
business logic
  minimizing in client 4-3

C

caching data
  to improve performance 4-4, 4-5
Calling
  conventional subroutine from NatStar TPR 5-8, 5-11, 5-58
changing
  Workspace 5-70
client
  definition g-1
client application
  definition g-1
  developing in NatStar C-1
  minimizing changes 4-3
client system
  definition g-1
client workstation
  definition g-1
client/server
  application (definition) 2-2
  application development 1-1, 1-3
  application development (step-by-step) 4-13
  architecture concepts 2-2
  creating configuration 2-5
  implementing 2-8
  three-tier 2-6
  two-tier 2-6
client/server application
  designing 4-1
  partitioning logic E-1
client/server interaction
  minimizing 4-4
CLISERVICES
  using entries to locate a server 5-6
COBOL
  Code Generation 5-90
  compiling code 5-94
  examples 9-2
  libraries 5-74
  linking 5-98
  using subroutines from NCL 9-1
COBOL-85
  avoiding recursion in NCL 5-12
  avoiding reserved words in NCL 5-12

Code Generation
C language 5-103
COBOL-85 5-90
NatStar generator COBOL output files 5-92
NCL 5-103, F-1
other than for NatStar TPR 5-88
pathnames in NatStar for COBOL D-7
server code 4-16, 5-70
commitment unit
data-integrity considerations 4-6
on GCOS 7 4-8
transaction generated 5-10
communication services
definition g-1
compiling
application F-3
client source 4-17, 5-103
COBOL code 5-94
TPR code 5-93
configuration
creating client/server 2-5
using files 5-6
Configurations
IM 5-78
Main 5-76
PM 5-79
Target 5-77
connection template G-8
conventional
definition g-2
conventional code
definition 2-14
simulating in NCL 5-101
conventional subroutine
calling from NatStar TPR 5-8, 5-11, 5-58
conventional TPR
definition 2-14
using NatStar-generated code 2-16
creating
Language Target Type 5-72
User Interface Target Type 5-73
Workspace 5-70

D

data transfers
minimizing number 4-4
minimizing volume 4-5
Data Truncation
handling 5-51
database access
from NATSTAR/TPR 2-15
isolating from business logic 4-3
database updates
against multiple databases 4-9
avoiding lost 4-8
database Wrapper
using 4-3
DATA-TYPE conversion
using COBOL MOVE statements 5-50
using NSCBL_CONVxxx 5-51
DATA-TYPE incompatibility
causes 5-10, 5-48
handling 5-10, 5-47
when calling conventional subroutine 5-11, 5-59
DEBUG BOX
not checked for code generation 5-90, C-18, F-2
Debugging
in NCL 4-16, 5-101
methods 4-17, 5-101
using NatStar development workstation 5-101
using PCF 5-102
using simulated GCOS 7 logic 5-101
Developers
definition g-2
using this document 1-2

E

error codes
NS02OP7G H-3
NSCOM Light H-3
Socket Interface I-1
TP H-1
WSA_INVALID_HANDLE I-9
WSA_INVALID_PARAMETER I-9

error codes (continued)
WSA_IO_INCOMPLETE I-10
WSA_IO_PENDING I-10
WSA_NOT_ENOUGH_MEMORY I-10
WSA_OPERATION_ABORTED I-13
WSAEADDRINUSE I-1
WSAEADDRNOTAVAIL I-2
WSAEALREADY I-2
WSAECCESS I-1
WSAECONNABORTED I-2
WSAECONNREFUSED I-3
WSAECONNRESET I-3
WSAEDESTADDRREQ I-3
WSAEDISCON I-13
WSAEFAULT I-3
WSAEFNOSUPPORT I-2
WSAEHOSTDOWN I-4
WSAEHOSTUNREACH I-4
WSAEINPROGRESS I-4
WSAEINTR I-4
WSAEINVAL I-4
WSAEISCONN I-5
WSAEMFILE I-5
WSAEMSGSIZE I-5
WSAENETDOWN I-5
WSAENETRESET I-5
WSAENETUNREACH I-6
WSAENOBUFS I-6
WSAENOPROTOOPT I-6
WSAENOTCONN I-6
WSAENOTSOCK I-6
WSAEOPNOTSUPP I-7
WSAEPFNOSUPPORT I-7
WSAEPROCLIM I-7
WSAEPROTONOSUPPORT I-7
WSAEPROTOTYPE I-8
WSAESHUTDOWN I-8
WSAESOCKTNOSUPPORT I-8
WSAETIMEDOUT I-8
WSAEWOULDBLOCK I-9
WSAHOST_NOT_FOUND I-9
WSAINVALIDPROCTABLE I-10
WSAINVALIDPROVIDER I-10
WSANO_DATA I-11
WSANO_RECOVERY I-11
WSANOTINITIALISED I-11
error codes (continued)
WSAPROVIDERFAILEDINIT I-11
WSASYSCALLFAILURE I-12
WSASYSNOTREADY I-12
WSATRY_AGAIN I-12
WSAVERNOTSUPPORTED I-12
Error Popup
using for diagnostics 5-63, E-3
Errors
handling 5-11, 5-61, C-15
when generating NCL 5-91
Examples
COBOL 9-2
compiling COBOL-85 5-97
data conversion using COBOL MOVE statements 5-50
data conversion using NSCBL_CONVTOF 5-54
using native code 5-57
using packed-decimal data type in NCL 5-45
extendibility
ensuring 4-2

F

fat client
definition 2-6
features of NatStar 1-3
Function
defining as Local or Remote 5-87
definition of term in NCL 2-7, 5-3
Functions
NSTC_CONNECT 8-5
NSTC_DEFINE_VIRTUAL_NODE 8-3
NSTC_DISCONNECT 8-6

G

generating the TDS F-5
GUI
definition g-2

I

IM
Configuration 5-78
Inline Code
see Native Code 5-11
installing
Services 5-74
instruction
defining an instruction in NCL B-4
defining as Local or Remote 5-87
definition of term in NCL 2-7, 5-3
integrity control
limitations 4-9

J

JCL
example JCL in NSCOM7 files 5-95
for compiling COBOL-85 5-97

L

LAN
performance implications 4-5
Language
Target Type 5-72
launching the TDS F-5
Library
associating client and server libraries C-11
creating a library in NatStar B-1
guidelines for using NatStar libraries 5-3
one or more instructions or functions 5-8
relationship of libraries to NatStar TPRs 5-4
using in NCL B-1
using to call a conventional subroutine 5-60
linking
application F-5
client code 4-17, 5-103
NatStar TPR code 5-98
NatStar/TPR code 4-17
lost updates
avoiding 4-8

M

Main Configuration 5-76
MAINTPR
use as TPR Driver 5-8
middleware
definition g-2

N

Native Code
disadvantage of using 5-56
using 5-11, 5-56
NatStar
benefits 1-3
definition 1-1, 1-3, g-2
features 1-3
TP error codes H-1
TPR (definition) 2-14
using 1-2
using independently 2-16
NatStar communication services
definition 1-5, g-2
using 2-15
using stubs 2-3
NatStar development workstation
definition g-2
required software 3-2
using 1-3, 5-3
NatStar TPR
COBOL stub associated with 5-8
compiling 5-93
contents of input message 5-8
differences from conventional TPRs 5-7
linking 5-98
processing flow 5-7
structure 5-7
topics to consider when developing 5-10
using one or more stubs/procedures 5-8
NCL
defining NatStar TPR logic 5-10
libraries 5-75
sequence listings G-1
using COBOL subroutines 9-1
using to define client logic C-13
using to define INIT event for client E-1

NCL (NatStar Control Language)
definition 1-3
network
optimizing performance 4-5
networking services
definition g-2
NS02OP7G
API 8-3
communication driver 8-1
error codes H-3
NSCBL_CONVTOC
using 5-51
NSCBL_CONVTOF
using 5-46, 5-51
NSCOM Light
error codes H-3
getting started 8-1
NSCOM.INI file
activating NSCOM Log File for error analysis 5-68
using to locate server 5-6
NSCOM7
definition 1-5, g-2
using 2-15
NSCOM7 code
definition 2-14
NSCOM8 Validation Package
used as example A-3
NSTC_CONNECT 8-5
NSTC_DEFINE_VIRTUAL_NODE 8-3
NSTC_DISCONNECT 8-6

O

Oracle support 7-1
ORB support 7-1

P

PACKED-DECIMAL
limitations on use 5-44
support in NatStar COBOL code generator 5-44
PACKED-DECIMAL DATA TYPE
using in NCL 5-10
partitioning
client/server logic 1-5, 2-5, 4-15
client/server NCL 1-4, 4-16
for extendibility 4-3
PCF 5-102
performance
achieving satisfactory 4-4
PM
Configuration 5-79

R

Recursion
avoiding in NCL that will be generated in COBOL 5-12
required software
NatStar development workstation 3-2
rollback
limitations in client/server applications 4-7
RPC
accessing Oracle database 7-2
automatic code generation 1-5
definition g-3
stubs 2-3
using 2-3, 2-15

S

SCB
definition g-3
segments
mapping NCL segments to COBOL data types 5-48
SEND/RECEIVE
restrictions in NatStar TPRs 5-60
separate libraries for client and server code B-1
server
definition g-3
server application
definition g-3
Server Configurations 5-75
server system
definition g-3

service system
definition g-3
Services
installing 5-74
Socket Interface
error codes I-1
SRVNTFND (Service Not Found) SERVICE
using to locate a server 5-6
SRVUFAS library G-1
Stubs
automatic generation 2-3, F-1
server 5-8
system administrator
definition g-3
using this document 1-2

T

Target
Configuration 5-77
Target Types
Language 5-72
User Interface 5-73
TDS
generation F-5
launching F-5
using in Bull/NatStar 5-4
Testing
methods 5-101
TESTUFAS library G-4
thin client
definition 2-6
THINGS_CLEAR_ERROR
using 5-65
THINGS_GET_DIAGNOSTIC
using 5-65
THINGS_GET_ERRMSG
using 5-65, C-15
THINGS_GET_ERROR FUNCTION
using 5-65, C-15
THINGS_GET_ERROR instruction
using 5-65
THINGS_SET_DEFAULT_SERVER
using 5-5, 5-88, E-1
three-tier
client/server 2-6
TP error codes H-1
TPR
definition of conventional TPR 2-14, g-2
definition of NatStar/TPR 2-14
TPR code
compiling 5-93
TPR Driver
automatic generation F-1
MAINTPR 5-8
transaction message
contents of input 5-8
passing as parameters 5-8
Truncation
data handling 5-51
two-tier
client/server 2-6

U

UNIX workstation
as client system g-1
as NatStar development workstation g-2
updates
avoiding lost 4-8
to multiple databases 4-9
User Interface
Target Type 5-73
user presentation
avoiding user confusion 4-7
minimizing changes 4-3

W

WAN
performance implications 4-5
Workspace
changing settings 5-70
WSA_INVALID_HANDLE error code I-9
WSA_INVALID_PARAMETER error code I-9
WSA_IO_INCOMPLETE error code I-10
WSA_IO_PENDING error code I-10
WSA_NOT_ENOUGH_MEMORY error code I-10

WSA_OPERATION_ABORTED error code I-13
WSAEADDRINUSE error code I-1
WSAEADDRNOTAVAIL error code I-2
WSAEAFNOSUPPORT error code I-2
WSAEALREADY error code I-2
WSAECCESS error code I-1
WSAECONNABORTED error code I-2
WSAECONNREFUSED error code I-3
WSAECONNRESET error code I-3
WSAEDESTADDRREQ error code I-3
WSAEDISCON error code I-13
WSAEFAULT error code I-3
WSAEHOSTDOWN error code I-4
WSAEHOSTUNREACH error code I-4
WSAEINPROGRESS error code I-4
WSAEINTR error code I-4
WSAEINVAL error code I-4
WSAEISCONN error code I-5
WSAEMFILE error code I-5
WSAEMSGSIZE error code I-5
WSAENETDOWN error code I-5
WSAENETRESET error code I-5
WSAENETUNREACH error code I-6
WSAENOBUFS error code I-6
WSAENOPROTOOPT error code I-6
WSAENOTCONN error code I-6
WSAENOTSOCK error code I-6
WSAEOPNOTSUPP error code I-7
WSAEPFNOSUPPORT error code I-7
WSAEPROCLIM error code I-7
WSAEPROTONOSUPPORT error code I-7
WSAEPROTOTYPE error code I-8
WSAESHUTDOWN error code I-8
WSAESOCKTNOSUPPORT error code I-8
WSAETIMEDOUT error code I-8
WSAEWOULDBLOCK error code I-9
WSAHOST_NOT_FOUND error code I-9
WSAINVALIDPROCTABLE error code I-10
WSAINVALIDPROVIDER error code I-10
WSANO_DATA error code I-11
WSANO_RECOVERY error code I-11
WSANOTINITIALISED error code I-11
WSAPROVIDERFAILEDINIT error code I-11
WSASYSCALLFAILURE error code I-12
WSASYSNOTREADY error code I-12
WSATRY_AGAIN error code I-12
WSAVERNOTSUPPORTED error code I-12

X

X.25 WAN
  performance implications 4-5